Selective Audio Filtering for Enabling Acoustic Intelligence in Mobile, Embedded, and Cyber-Physical Systems

Stephen Xia

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy under the Executive Committee of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2022
Abstract
Selective Audio Filtering for Enabling Acoustic Intelligence in Mobile, Embedded, and Cyber-Physical Systems
Stephen Xia

We are seeing a revolution in computing and artificial intelligence; intelligent machines have become ingrained in and improved every aspect of our lives. Despite the increasing number of intelligent devices and breakthroughs in artificial intelligence, we have yet to achieve truly intelligent environments. Audio is one of the most common sensing and actuation modalities used in intelligent devices. In this thesis, we focus on how we can more robustly integrate audio intelligence into a wide array of resource-constrained platforms that enable more intelligent environments. We present systems and methods for adaptive audio filtering that enables us to more robustly embed acoustic intelligence into a wide range of real-time and resource-constrained mobile, embedded, and cyber-physical systems that are adaptable to a wide range of different applications, environments, and scenarios.

First, we introduce methods for embedding audio intelligence into wearables, like headsets and helmets, to improve pedestrian safety in urban environments by using sound to detect vehicles, localize vehicles, and alert pedestrians well in advance to give them enough time to avoid a collision. We create a segmented architecture and data processing pipeline that partitions computation between embedded front-end platform and the smartphone platform. The embedded front-end hardware platform consists of a microcontroller and commercial-off-the-shelf (COTS) components embedded into a headset and samples audio from an array of four MEMS microphones. Our em-
bedded front-end platform computes a series of spatiotemporal features used to localize vehicles: relative delay, relative power, and zero crossing rate. These features are computed in the embedded front-end headset platform and transmitted wirelessly to the smartphone platform because there is not enough bandwidth to transmit more than two channels of raw audio with low latency using standard wireless communication protocols, like Bluetooth Low-Energy. The smartphone platform runs machine learning algorithms to detect vehicles, localize vehicles, and alert pedestrians. To help reduce power consumption, we integrate an application-specific integrated circuit into our embedded front-end platform and create a new localization algorithm called \textit{angle via polygonal regression} (AvPR) that combines the physics of audio waves, the geometry of a microphone array, and a data-driven training and calibration process that enables us to estimate the high resolution direction of the vehicle while being robust to noise resulting from movements in the microphone array as we walk the streets.

Second, we explore the challenges in adapting our platforms for pedestrian safety to more general and noisier scenarios, namely \textit{construction worker safety} sounds of nearby power tools and machinery that are orders of magnitude greater than that of a distant vehicle. We introduce an \textit{adaptive noise filtering architecture} that allows workers to filter out construction tool sounds and reveal low-energy vehicle sounds to better detect them. Our architecture combines the strengths of both the physics of audio waves and data-driven methods to more robustly filter out construction sounds while being able to run on a resource-limited mobile and embedded platform. In our adaptive filtering architecture, we introduce and incorporate a data-driven filtering algorithm, called \textit{probabilistic template matching} (PTM), that leverages pre-trained statistical models of construction tools to perform content-based filtering. We demonstrate improvements that our adaptive filtering architecture brings to our audio-based urban safety wearable in real construction site scenarios and against state-of-art audio filtering algorithms, while having a minimal impact on the power consumption and latency of the overall system. We also explore how these methods can be used to improve audio privacy and remove privacy-sensitive speech from applications that have no need to detect and analyze speech.
Finally, we introduce a common selective audio filtering platform that builds upon our adaptive filtering architecture for a wide range of real-time mobile, embedded, and cyber-physical applications. Our architecture can account for a wide range of different sounds, model types, and signal representations by integrating an algorithm we present called content-informed beamforming (CIBF). CIBF combines traditional beamforming (spatial filtering using the physics of audio waves) with data-driven machine learning sound detectors and models that developers may already create for their own applications to enhance and filter out specified sounds and noises. Alternatively, developers can also select sounds and models from a library we provide. We demonstrate how our selective filtering architecture can improve the detection of specific target sounds and filter out noises in a wide range of application scenarios. Additionally, through two case studies, we demonstrate how our selective filtering architecture can easily integrate into and improve the performance of real mobile and embedded applications over existing state-of-art solutions, while having minimal impact on latency and power consumption. Ultimately, this selective filtering architecture enables developers and engineers to more easily embed robust audio intelligence into common objects found around us and resource-constrained systems to create more intelligent environments.
**Table of Contents**

Acknowledgments .............................................. xii

Dedication ..................................................... xiv

Chapter 1: Introduction .................................... 1
    1.1 Main Components and Contributions ................. 4
        1.1.1 Embedded Acoustic Intelligence for Improving Pedestrian Safety .... 5
        1.1.2 Adaptive Noise Filtering for Improving Construction Worker Safety and Audio Privacy .................. 6
        1.1.3 A Common Selective Audio Filtering Framework for Mobile, Embedded, and Cyber-Physical Systems .................... 6
    1.2 Contributions to Literature .......................... 7

Chapter 2: Embedded Acoustic Intelligence for Improving Pedestrian Safety ........ 9
    2.1 Related Work ........................................ 12
    2.2 Studying the Problem ................................ 14
        2.2.1 Recording Specifications .......................... 14
        2.2.2 Presence of a Car ............................... 16
        2.2.3 Direction of a Car ............................... 18
        2.2.4 Distance of a Car ............................... 19
    2.3 Overview of PAWS .................................... 20
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.1 Front-End Hardware</td>
<td>22</td>
</tr>
<tr>
<td>2.3.2 Front-End Signal Processing</td>
<td>23</td>
</tr>
<tr>
<td>2.3.3 Smartphone Data Processing</td>
<td>25</td>
</tr>
<tr>
<td>2.4 PAWS Low-Energy</td>
<td>30</td>
</tr>
<tr>
<td>2.4.1 Reducing Power through a Custom ASIC</td>
<td>30</td>
</tr>
<tr>
<td>2.4.2 Resolving Constraints Imposed by Utilizing the ASIC</td>
<td>31</td>
</tr>
<tr>
<td>2.4.3 Angle via Polygonal Regression</td>
<td>32</td>
</tr>
<tr>
<td>2.4.4 PAWS Low-Energy System Architecture</td>
<td>36</td>
</tr>
<tr>
<td>2.5 Platform Evaluation</td>
<td>37</td>
</tr>
<tr>
<td>2.5.1 Real-Time Performance</td>
<td>37</td>
</tr>
<tr>
<td>2.5.2 Power Consumption and Price Breakdown</td>
<td>43</td>
</tr>
<tr>
<td>2.6 Real-World Evaluation</td>
<td>44</td>
</tr>
<tr>
<td>2.6.1 Experimental Setup</td>
<td>45</td>
</tr>
<tr>
<td>2.6.2 Results</td>
<td>47</td>
</tr>
<tr>
<td>2.6.3 Limitations and Future Work</td>
<td>51</td>
</tr>
<tr>
<td>Chapter 3: Adaptive Acoustic Noise Filtering for Improving Construction Worker Safety</td>
<td>55</td>
</tr>
<tr>
<td>3.1 Related Works</td>
<td>57</td>
</tr>
<tr>
<td>3.2 Overview of Source Separation and Selective Noise Filtering</td>
<td>58</td>
</tr>
<tr>
<td>3.2.1 Spatial Source Separation</td>
<td>58</td>
</tr>
<tr>
<td>3.2.2 Content-Based Separation</td>
<td>59</td>
</tr>
<tr>
<td>3.3 CSafe Filtering Architecture</td>
<td>61</td>
</tr>
<tr>
<td>3.3.1 Content-Based Separation: Probabilistic Template Matching</td>
<td>62</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Examples of environments and application scenarios that a combined selective filtering framework could benefit, including: i) home environments: enhancing command phrases while filtering loud music that could make detection more difficult, ii) urban safety: enhancing vehicle sounds and filtering out construction sounds that make detection more difficult, iii) audio privacy: for example, filtering out privacy sensitive speech in a mobile sleep monitoring application that records and analyzes only breathing and sleep sounds, iv) entertainment: for example, enhancing the sound of animals at a zoo while filtering wind and other noises, and many more not shown. ................................................................. 2

2.1 An inattentive pedestrian wearing a PAWS headset, and a screen shot of the PAWS application user interface. ................................................................. 10

2.2 Validation system: reference mannequin with eight MEMS microphones and data acquisition board in a low-noise experimental setup. .............................. 15

2.3 Spectrogram of one of the recordings from the controlled environment. The car was approaching the mannequin at 25mph. ............................................... 16

2.4 Distribution of honks and other types of sounds in a 2D feature space. ................. 17

2.5 Normalized relative delays of the microphones for left and right honks. ............. 18

2.6 Relative delay versus time of a car driving past a mannequin from its left to right. . 19

2.7 The maximum cepstral coefficient follows a trend when an approaching car is within about 30m from an observer. ....................................................... 20

2.8 A block diagram of PAWS. The components highlighted in red are portions of the system that are enhanced and modified during the second phase of development to create PAWS Low-Energy, which is discussed in Section 2.4. .......................... 21
2.9  (Left) Teardown of the PAWS headset; the front-end hardware is exposed inside the left ear housing. (Right) Close up of the PCB that comprises the PAWS front-end hardware. .......................................................... 22

2.10 PAWS Smartphone data processing. The components highlighted in red are portions of the system that are improved upon in PAWS Low-Energy, which is discussed in Section 2.4. .......................................................... 25

2.11 Illustration of the basic idea of non-uniform binning of spectral energy in NBIP, introduced by our collaborators1. .......................................................... 26

2.12 (a) The NBIP feature vector for car tire and engine sounds are designed to maximize their dissimilarity from non-car street noises like human chatter, human motions, machine sounds, and loud music. (b) standard MFCC features are not as effective in separating the two classes as our proposed NBIP features. .......................................................... 28

2.13 Plot of relative delay points from playing white noise in eight directions around the user. .......................................................... 35

2.14 AvPR mean and variance interpolation between sampled directions to create the polygon model used to estimate all other directions. .......................................................... 36

2.15 Estimating a new direction with AvPR. Blue dashed lines are example cases of computing the shortest distance from the new point to a segment. The solid black line denotes the path from the new point to the point on the polygon that is closest to the new point. .......................................................... 37

2.16 Pipeline of the PAWS and PAWS Low-Energy feature extraction process. In the PAWS flow, the “Features Calc.” block represents all the operations involved in the features extraction, and “TX” represents the UART communication between the MCU and BLE module. In the PAWS Low-Energy flow, “Extract” refers to the BLE module reading relative delays from the ASIC. .......................................................... 39

2.17 Block diagram of the test setup for the latency between the features from the front-end hardware and the smartphone. .......................................................... 41

2.18 Histogram of PAWS front-end hardware to smartphone latency acquired with the test setup shown in Figure 2.17. .......................................................... 41

2.19 Histogram of PAWS Low-Energy front-end hardware to smartphone latency acquired with the test setup shown in Figure 2.17. We see that the mean latency is shorter than PAWS. .......................................................... 42

2.20 Execution times of various components of the PAWS and PAWS Low-Energy smartphone app. .......................................................... 42
2.21 Experiment scenario in campus street. ................................. 45
2.22 Car detection performance. .................................................. 48
2.23 Metro area car detection confusion matrix. ............................... 48
2.24 Estimated direction vs. true direction estimated via AvPR. ............... 49
2.25 Estimated direction vs. true direction estimated via triangulation. ........... 50
2.26 Combined accuracy for PAWS directions prediction. ....................... 50
2.27 Car localization accuracy. ..................................................... 51
2.28 Confusion matrices for distance estimation. ................................ 52
2.29 Estimated distances for cars within 30m are on average 2.8m off of actual distances. 52

3.1 (Left) The CSafe embedded platform, consisting of an array of four microphones integrated into a feature extraction system introduced in [8]. Two microphones are shown (the front and right microphones), while the other two are on the back and left side of the helmet. (Right) Screenshot of CSafe’s smartphone system. 56

3.2 CSafe’s full source separation architecture combining PTM (content-based separation) and spatial separation. ................................. 61

3.3 (a) Plot demonstrating the extraction process of PTM with a highly confident "template". We see that if we are more confident in the frequency component of our sound source (lower variance), PTM will extract coefficients or generated "predicted values" closer to the mean of the model template. b) However, if the model of the frequency component has higher variance (e.g. we are less confident in value of the frequency component), PTM will extract coefficients further away from the mean of the template. .................................................. 65

3.4 Examples of filtering results of a vehicle sound mixed over-the-air in the real world with a jackhammer sound. We played the vehicle sound (3.4(a)) and jackhammer sound (3.4(b)) through two different speakers and recorded the mixture with a 4-channel uniform circular microphone array to generate the mixed waveform (3.4(c)). We see from visually inspecting the filtered waveforms that the MM-DenseLSTM neural network approach (3.4(d)) was not able to remove as much of the jackhammer noise as PTM. (3.4(e)). ................................. 66
3.5 a) (Left) A wider and less directional filter. (Right) More energy of the original signal is preserved. b) (Left) A more directional filter. (Right) More energy from other directions are filtered out at the cost of greater signal distortion. The signal shown in both scenarios were recorded with a four microphone uniform circular array by playing a vehicle and jackhammer sound with separate speakers on repeat. The filtered signals (right plots) were generated by applying filters (left plots) to diminish the construction tool sound. .......................... 71

3.6 Experiment setup at a construction site. ................................. 73

3.7 The two plots show a recording of an approaching vehicle in presence of a power tool sound when: a) No noise filtering is applied, and b) CSafe’s noise filtering architecture is applied. The green segments highlight the ground truth for when a vehicle is present and the red arrows highlight the segments where the vehicle detector detects the presence of a vehicle. We see in this example, that the vehicle detector can detect the presence of vehicles earlier in each segment after applying our noise filtering architecture. ................................. 77

3.8 Average latency of vehicle detection of our vehicle detector without applying noise filtering and after introducing noise filtering. We see that CSafe’s noise filtering architecture is able to reduce detection latency from 42.5ms, without any filtering, down to 16ms, which is a greater reduction than any of the other methods tested. . . 79

3.9 CSafe’s full system architecture spanning the embedded and smartphone sub-platforms. 79

3.10 Breakdown of latency of each component of CSafe. CSafe completes one cycle of data sampling and computation in 236ms, which is on par with the reaction time of an average person. .......................... 82

3.11 PAMS architecture and data pipeline. The filter coefficients, \( \alpha_i \), are updated based on the speech templates provided by a Gaussian Mixture noise model and applied in the next window. ................................. 88

3.12 PAMS enhanced sleep monitoring system architecture and data pipeline . . . . . 88

3.13 Confusion matrix metrics for the sleep event detectors of Sleep as Android and PAMS. .......................... 90

4.1 Pipeline detailing the CIBF enhancement and filtering optimization process. The forward pass shows the typical steps of evaluating the presence of a sound. The CIBF optimization process is highlighted in red (backward pass), highlighting the three primary steps. Each step is directly tied to one part of the detection pipeline, where connections are highlighted using black dashed arrows. .......................... 101
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>AvA’s System Architecture.</td>
<td>105</td>
</tr>
<tr>
<td>4.3</td>
<td>AvA-enhanced urban safety wearable architecture. The embedded hardware platform is borrowed from [8]. AvA directly uses the results from the vehicle detector to determine when to alert the user.</td>
<td>115</td>
</tr>
<tr>
<td>4.4</td>
<td>AvA-enhanced platform for filtering out speech and preserving privacy in mobile applications. Unlike in urban safety, audio privacy applications save raw breathing sounds. In the AvA-enhanced systems, we save the filtered signals rather than the raw audio to preserve privacy.</td>
<td>117</td>
</tr>
<tr>
<td>4.5</td>
<td>BuMA’s system architecture.</td>
<td>123</td>
</tr>
<tr>
<td>4.6</td>
<td>Confusion Matrix of multi-class sound detector</td>
<td>124</td>
</tr>
<tr>
<td>4.7</td>
<td>Data collection set up. We placed BuMA either to the side or above the mattress, next to where a person would sleep. Then, we played an interfering noise (construction, speech, or music) from a speaker. In this figure, the distance between speaker and the person’s head is 150 cm, and BuMA is 30 cm away right above the person lying down on the mattress in the supine position.</td>
<td>126</td>
</tr>
<tr>
<td>4.8</td>
<td>Accuracy vs. distance BuMA is away from the head of the person, when BuMA is above the mattress (top) and to the side of the mattress (bottom). The average SNR is listed above each distance.</td>
<td>127</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Power Consumption and Price Breakdown of PAWS and PAWS Low-Energy (PAWS Low-Energy Total in Bold) ........................................... 43

2.2 Summary of Deployment Events. .............................................. 46

3.1 Confusion matrix metrics of vehicle detectors under different filtering schemes in a construction site environment. ........................................... 75

3.2 True positive (recall) for vehicle detection broken down by SNR and tool type of the environment. .............................................. 75

3.3 Confusion matrix metrics comparing both modes of CSafe with a state-of-art pedestrian safety system [8] ........................................... 84

3.4 Localization error comparison between CSafe and a state-of-art pedestrian safety system, PAWS [8] ........................................... 84

3.5 Speech recognition accuracy of recorded clips from PAMS and Sleep as Android [102]. This table lists the proportion of words correctly identified in the recorded clips, the proportion of words incorrectly identified, and the proportion of words that were not even detected by Google Speech to Text [105] ........................................... 90

4.1 Summary of evaluation scenarios and different configurations of AvA. .............. 107

4.2 Target and noise detection performance in scenario 1 (target: crying and sobbing + noise: construction). ........................................... 109

4.3 Target and noise detection performance in scenario 2 (target: dog + noise: vehicles). 109

4.4 Target and noise detection performance in scenario 3 (target: piano + noise: speech-noise). ........................................... 110
4.5 Target and noise detection performance in scenario 4 (target: wild animals + noise: wind). ................................................................. 110

4.6 Performance metrics of vehicle detection of AvA compared to other state-of-art acoustic-based urban safety wearables, while user is having a phone conversation. . 115

4.7 Localization error comparison between AvA and other state-of-art acoustic-based urban safety wearables, while user is having a phone conversation. ............ 115

4.8 Power consumption and latency comparison between an AvA-enhanced wearable and other state-of-art urban safety wearables. The total latency is the time it takes for each system to process one window of audio. The power consumption shows the current draw from each embedded platform powered by a 3.3V battery. ......... 116

4.9 Proportion of words correctly identified, incorrectly identified, and undetected by Google Speech-to-Text. A lower rate of correctly identified words correlates to a more privacy-aware system. .................................................. 118

4.10 Performance metrics for sleep breathing detection. ................................. 119

4.11 Speech intelligibility of the AvA enhanced command phrase mobile application in scenarios where a command phrase was present and not present. .............. 120
Acknowledgements

During my time at Columbia University, I have been extremely fortunate to have support from a cast of amazing and wonderful individuals. I would first like to express sincere gratitude and thank my advisor, Professor Xiaofan (Fred) Jiang, who has worked tirelessly to enable my fruitful journey. All my work and everything that I have learned along the way is all because of his insight, wisdom, and guidance. Professor Jiang has not only nurtured me as a researcher, but has also contributed to my growth and maturity as a decent human being.

I am also grateful to have opportunities to work with Professor Peter Kinget and Professor Shahriar Nirjon, who have provided their great insight as researchers from other fields and perspectives. I would also like to thank my amazing collaborators, outside of Columbia, who I have shared many conversation and discussions about research and life: Bashima Islam, Md Tamzeed Islam and Shibo Zhang. I would also like give thanks to my former and current colleagues, who have been supporting my research through thick and thin: Rishikanth Chandrasekaran, Daniel de Godoy Peixoto, Jingping Nie, Yanchen Liu, Minghui (Scott) Zhao, Kaiyuan Hou, Asmita Goyanka, Laixi Shi, Ao Liang, Chenye Yang, Hengjiu Kang, Ji Jia, Hugo Meyer, Avik Dhupar, Yigong Hu, Yan Lu, Xiaoqi (Danny) Chen, Jordan Misael Vega, and Andrew Sonta. I would also like to extend my thanks to Professor Romit Roy Choudhury and Professor James Anderson, who have participated on my defense committee.

Along the way, I have met and made many wonderful and amazing friends who have given immense support on my journey: Lei Lei, Mark Barbet, Joe Lee, Ankur Nipane, Anjaly Rajendran, Abhinandan Borah, Han Wang, Shaoru Chen, Jane Zhang, Tingjun Chen, Qinghui Meng, Yian Yu,
Soonhee Moon, Tetsuhiro Harimoto, Tiffany Chien, and Angela Hu. I am extremely fortunate and
grateful to have met Peter Wei, who has walked down the same road as me; both of us started
down our path towards a Ph.D. the same time and have shared both the ups and downs during our
time here. I am also extremely thankful for Tingkai Liu and Zichao (Jack) Wang, who I met as
undergraduates, and have inspired me to become a researcher.

Finally, I would like to thank my family and my parents, who have shown unconditional love
and support throughout this journey and throughout my entire life.

Financial Support: The research in this thesis was partially supported by the National Sci-
ence Foundation under Grant Numbers CNS-1704899, CNS-1815274, CNS-11943396, and CNS-
1837022. The views and conclusions contained here are those of the authors and should not be
interpreted as necessarily representing the official policies or endorsements, either expressed or
implied, of Columbia University, NSF, or the U.S. Government or any of its agencies.
To my mother and father, Yili Wang and Yunqing Xia.
Chapter 1: Introduction

We are seeing a revolution in computing and artificial intelligence; intelligent machines have become ingrained in and improved every aspect of our lives. Despite the increasing number of intelligent devices and breakthroughs in artificial intelligence, we have yet to achieve truly intelligent environments. Many of our cities, homes, and offices are becoming “smarter” primarily because there is an ever-growing number of smart devices tailored to specific tasks being released every year. However, these devices are still not being utilized jointly to their maximum potential [1]. There are a wide range of different sensing modalities used in intelligent devices including, but not limited to vision, touch, temperature, and light. Audio is one of the most common sensing and actuation modalities used in intelligent devices. In this thesis, we focus on how we can more robustly integrate audio intelligence into a wide array of resource-constrained platforms that enable more intelligent environments.

We explore methods to embed audio intelligence into wearables, sensor networks, and common objects found all around us to enable a wide range of intelligent audio-based applications. The number of smart devices in our environments is growing rapidly. In the home space, the market for home-based smart devices is expected to grow from USD 102 billion in 2021 to USD 182 billion by 2025 [2]. We are also expecting to see a similar trend in the wearables market, which is projected to grow from USD 116.2 billion in 2021 to USD 265.4 billion by 2026 [3]. Many of these devices use audio as a means to communicate with people, as well as sense and understand the physical world. In 2020, the smart personal audio device grew by 20% [4]. From 2019 to 2025, the smart speaker market, alone, is expected to grow from USD 7.1 billion to USD 15.6 billion [5]. This drastic increase in smart devices and audio devices opens many opportunities, as well as challenges, to make our environments more intelligent and improve our quality of life.

Audio-based smart devices use microphones to listen to the environment and analyze sounds
Figure 1.1: Examples of environments and application scenarios that a combined selective filtering framework could benefit, including: i) home environments: enhancing command phrases while filtering loud music that could make detection more difficult, ii) urban safety: enhancing vehicle sounds and filtering out construction sounds that make detection more difficult, iii) audio privacy: for example, filtering out privacy sensitive speech in a mobile sleep monitoring application that records and analyzes only breathing and sleep sounds, iv) entertainment: for example, enhancing the sound of animals at a zoo while filtering wind and other noises, and many more not shown.

using machine learning and signal processing algorithms to execute tasks and provide services. Often, the system needs to detect and analyze specific sound(s), while disregarding all others. This
raises two major concerns in embedding acoustic intelligence into resource-constrained mobile, embedded, and cyber-physical platforms. First, there are many instances where the target sound is difficult to detect and analyze because other noises in the environment can overpower the target sound. Second, many intelligent audio devices listen to the environment for long periods of time, where most of the time the target sound is not present. In these situations, our devices may be recording and listening to sounds in the environment that may pose privacy risks.

A combined filtering framework, where users can select specific sounds to keep and enhance or remove and filter may be extremely beneficial to a wide range of different applications. Examples of such applications are shown in Figure 1.1, which is by no means an exhaustive list. For example, in the home environment space, a smart assistant may have trouble detecting spoken commands if loud music or other noises are present. A filter that can enhance specific command phrases, while filtering out music, would be greatly enhance the usability of the home assistant. In the urban safety space, a selective filtering architecture would enable audio-based safety wearables to filter out loud construction noises that overpower vehicle sounds to better detect and localize oncoming vehicles. In the audio privacy space, a selective filtering architecture that can filter out speech, while retaining or enhancing other sounds in the environment (e.g., breathing sounds for sleep monitoring), would greatly improve the privacy of individuals. As a final example, in the entertainment space, a selective filtering architecture that can filter out noisy wind and enhance animal sounds in a zoo-based augmented or virtual reality wearable could greatly improve the user experience. There are an infinite number of scenarios where structured noises adversely affect the performance of acoustic detectors or pose a privacy risk, both of which are major challenges in realizing robust acoustic intelligence in many mobile, embedded, and cyber-physical applications.

In addition to challenges related to the detection performance of acoustic-based smart applications, there are a number of systems-based considerations that make this problem even more challenging. First, many smart-devices that are embedded into our homes, offices, and environments have limited computational resources. Second, many of these applications need to run in real-time. Because of these constraints, the methods we design need to be capable of running on
resource-constrained platforms like mobile phones and embedded microcontrollers; relying on off-loading computation to GPU-equipped servers or the cloud is not feasible in many scenarios due to latency.

We present systems and methods for adaptive audio filtering that enables us to more robustly embed acoustic intelligence into a wide range of real-time and resource-constrained mobile, embedded, and cyber-physical systems that are adaptable to a wide range of different applications, environments, and scenarios.

1.1 Main Components and Contributions

In this section, we present an overview of the main contributions of this thesis. We divide the contributions of this thesis into three main components. First, we introduce methods for embedding audio intelligence into wearables, like headsets and helmets, to improve pedestrian safety in urban environments by using sound to detect vehicles, localize vehicles, and alert pedestrians well in advance to give them enough time to avoid a collision. Second, we explore the challenges in adapting our platforms for pedestrian safety to more general and noisier scenarios, namely construction worker safety where there are sounds of power tools and machinery that are orders of magnitude greater than that of a typical vehicle. We introduce an adaptive noise filtering architecture that allows workers to filter out construction tool sounds and reveal low-energy vehicle sounds to better detect them. We also explore how these methods can be used to improve audio privacy and remove privacy-sensitive speech from applications that have no need to detect and analyze speech. Finally, we introduce a common selective audio filtering framework that builds upon our adaptive filtering architecture for a wide range of real-time mobile, embedded, and cyber-physical applications. This architecture enables developers to select, not only noises to filter out, but also sounds to keep and enhance for their unique applications. This architecture is also capable of supporting a wide range of different detector and sound model types, allowing us to more robustly integrate audio intelligence into a wide range of different mobile, embedded, and cyber-physical applications.
1.1.1 Embedded Acoustic Intelligence for Improving Pedestrian Safety

With the prevalence of smartphones, pedestrians and joggers today often walk or run while listening to music. Since they are deprived of their auditory senses that would have provided important cues to dangers, they are at a much greater risk of being hit by cars or other vehicles. In Chapter 2, we introduce an audio wearable for detecting vehicles, localizing vehicles, and alerting pedestrians in advance to prevent pedestrian-vehicle accidents. We embed a headset wearable with an array of microphones and an embedded front-end platform. We create a segmented architecture and data processing pipeline that partitions computation between embedded front-end platform and the smartphone platform. The embedded front-end hardware platform consists of a microcontroller and commercial-off-the-shelf (COTS) components and samples audio from an array of four MEMS microphones. Our embedded front-end platform computes a series of spatiotemporal features used to localize vehicles: relative delay, relative power, and zero crossing rate. These features are computed in the embedded front-end headset platform and transmitted wirelessly to the smartphone platform because there is not enough bandwidth to transmit more than two channels of raw audio with low latency using standard wireless communication protocols, like Bluetooth Low-Energy. Finally, the smartphone platform runs machine learning algorithms to detect vehicles, localize vehicles, and alert pedestrians.

To help reduce power consumption, we integrate an application-specific integrated circuit (ASIC effort led by Daniel de Godoy [6]) into our embedded front-end platform in the second phase of this work. However, because the ASIC computes only one specific feature that we use in localizing vehicles, we lose many of the spatio-temporal features we required to estimate the location of vehicles. As such, we create a new localization algorithm called angle via polygonal regression (AvPR) that combines the physics of audio waves, the geometry of a microphone array, and a data-driven training and calibration process that enables us to estimate the high resolution direction of the vehicle while being robust to noise resulting from movements in the microphone array as we walk the streets.
1.1.2 Adaptive Noise Filtering for Improving Construction Worker Safety and Audio Privacy

In Chapter 3, we adapt our pedestrian safety platform, introduced in Chapter 2, for improving construction worker safety. Because construction workers are using power tools and machinery that are orders of magnitude greater than that of oncoming vehicles, our audio wearable pedestrian safety platform had difficulty hearing and detecting vehicles. To address this challenge, we introduce an adaptive audio noise filtering architecture to filter out construction sounds, enabling us to better detect vehicle sounds. Our architecture combines the strengths of both the physics of audio waves and data-driven methods to more robustly filter out construction sounds while being able to run on a resource-limited mobile and embedded platform. In our adaptive filtering architecture, we introduce and incorporate a data-driven filtering algorithm, called probabilistic template matching (PTM), that leverages pre-trained statistical models of construction tools to perform content-based filtering. We demonstrate improvements that our adaptive filtering architecture brings to our audio-based urban safety wearable in real construction site scenarios and against state-of-art audio filtering algorithms, while having a minimal impact on the power consumption and latency of the overall system.

We also explore how our adaptive filtering architecture can be used to improve the privacy of smart home applications that use audio to analyze specific sounds in the environment (e.g., breathing sounds for monitoring sleep quality), but can also record other non-required privacy-sensitive signals (e.g., speech). In these applications, we show how our adaptive noise filtering architecture can be used to remove privacy-sensitive signals, like speech, while minimally impacting the performance of detecting target sounds.

1.1.3 A Common Selective Audio Filtering Framework for Mobile, Embedded, and Cyber-Physical Systems

In Chapter 4 we extended the adaptive filtering architecture introduced in Chapter 3 to a wide range of different applications and scenarios. Specifically, Chapter 3 gave us insight into how audio filtering could be used to improve the detection performance of vehicles by filtering out
construction sounds in a construction site, where we expect both vehicles and construction noises to be present. From this work, we realize that there are numerous applications, beyond urban safety, that could potentially benefit from an audio filtering architecture that enables developers and engineers to select specific sounds to enhance and keep (target sounds) while filtering out and removing other sounds they expect to interfere with their systems (noises). We accomplish this goal by introducing a **common selective audio filtering framework** that is adaptable to a wide range of different sounds, noises, and applications. Our architecture can account for a wide range of different sounds, model types, and signal representations by integrating an algorithm we create called **content-informed beamforming** (CIBF). CIBF combines traditional beamforming (spatial filtering using the physics of audio waves) with data-driven machine learning sound detectors and models that developers may have already created for their own applications to enhance and filter out specified sounds and noises. Alternatively, developers can also select sounds and models from a library we provide.

We demonstrate how our selective filtering architecture can improve the detection of specific target sounds and filter out noises in a wide range of application scenarios. Additionally, through two case studies, we demonstrate how our selective filtering architecture can easily integrate into and improve the performance of real mobile and embedded applications over existing state-of-art solutions, while having minimal impact on latency and power consumption.

### 1.2 Contributions to Literature

The audio wearable platform for improving pedestrian safety was published in the ACM/IEEE International Conference on Internet of Things Design and Implementation 2018 (IoTDI’18) [7] and the IEEE Internet of Things Journal 2019 (IoTJ’19) [8]. This research also received the Best Demo Runner Up Award (ACM SenSys 2016) [9], Best Demo Award (ACM/IEEE IoTDI’18) [10], Best Presentation Award (IEEE VNC’18), and Best App Runner Up Award (IEEE VNC’18) [11]. This work has also been featured by various news and media outlets including New York Post, IEEE Spectrum, Fast Company, Mashable, Gizmodo, The Telegraph, Engineering.com, India
Times, and IEEE Signal Processing Magazine. Our work on adaptive audio noise filtering for construction worker safety and audio privacy has been published in the ACM/IEEE International Conference on Information Processing in Sensor Networks 2021 (IPSN’21) [12] and the Second International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (AIChallengeIoT’20) [13]. The common selective audio filtering platform for mobile, embedded, and cyber-physical systems will appear later this year in the ACM/IEEE International Conference on Information Processing in Sensor Networks 2022 (IPSN’22).
Chapter 2: Embedded Acoustic Intelligence for Improving Pedestrian Safety

Smartphones have transformed our lifestyles dramatically, mostly for the better. Unfortunately, smartphone usage while walking has become a serious safety problem for many people in urban areas around the world. Pedestrians listening to music, texting, talking or otherwise absorbed in their phones are putting themselves at risk by tuning out the traffic around them [14], as reported by the Washington Post. Since a pedestrian is deprived of auditory input that would have provided important cues to dangers such as honks or noises from approaching cars, he or she is at a much greater risk of being involved in a traffic accident. We have seen a sharp increase in injuries and deaths from such incidents in recent years. According to a study by Injury Prevention and CNN, the number of serious injuries and deaths occurring to pedestrians who were walking with headphones has tripled in the last seven years in the United States [15]. This phenomenon affects cities globally, and is an important societal problem that we want to address by introducing advanced sensing techniques and intelligent wearable systems.

In this chapter, we present PAWS, a Pedestrian Audio Wearable System aimed for urban safety. The PAWS project is in collaboration with my colleagues both at Columbia University and the University of North Carolina at Chapel Hill. PAWS is a low-cost headset-based wearable platform that combines four MEMS microphones, signal processing and feature extraction electronics, and machine learning classifiers running on a smartphone to help detect and locate imminent dangers, such as approaching cars, and warn pedestrians in real-time. Figure 2.1 shows PAWS in action.

We take an audio approach to detect, localize, and alert users to oncoming vehicles in real-time and in a resource-limited wearable. Works that leverage sensors that provide more rich information about the environment, such as camera or LIDAR, have high computational and battery requirements, making it difficult to create a long-lasting battery-powered platform. Although audio does not provide as rich information about the environment as sensors like cameras, standard micro-
phones provide 360 degree coverage and require less computation to sample and extract features.

With newer smartphones equipped with multiple built-in microphones, it may be tempting to re-purpose those microphones in software to localize cars based on common localization techniques. However, these approaches require the user to constantly hold their phones steady and to not block the built-in microphone while walking [16][17]. Further, most built-in microphones are designed for voice and are often band-limited. These two limitations prevent the smartphone from capturing useful features produced by approaching cars in realistic urban environments.

This is a challenging problem as the battery-powered wearable platform needs to detect, identify, and localize approaching cars in real-time, process and compute large amounts of data in an energy and resource constrained system, and produce accurate results with minimal false positives and false negatives. For example, if a user’s reaction-time is 500ms, the system has 360ms to detect a 25mph car and alert the user when it is 10m away. This problem is further compounded by high levels of mixed noise, typical of realistic street conditions.

We address these challenges over two phases of design and development. In the first phase (PAWS), we develop a segmented architecture and data processing pipeline that partitions com-
putation into processing modules across a front-end hardware platform and a smartphone. The microcontroller-based front-end hardware platform consists of commercial off-the-shelf (COTS) components embedded into a standard headset and collects four channels of audio from four MEMS microphones that are strategically positioned on the headset. Temporal-spatial features such as relative delay, relative power, and zero-crossing rate are computed inside the front-end platform using the four channels and transmitted wirelessly to a smartphone. A fifth standard headset microphone is also connected to the audio input of the smartphone, and together with the data sent from the front-end platform, classifiers are trained and used to detect an approaching car and estimate its azimuth and distance from the user.

In the second phase of development (PAWS Low-Energy), we tackle the challenge of power consumption through the design and implementation of an application-specific integrated circuit to extract some of the computationally expensive features. We also develop new methods to increase the accuracy and granularity of our audio-based vehicle localization. We evaluate PAWS using both controlled experiments inside parking lots and real-world deployments on urban streets.

We summarize the major points of this chapter below:

• We incorporate a new acoustic feature, Non-Uniform Binned Integral Periodogram, which is designed to capture frequency domain characteristics of low-frequency noise-like sounds, such as the sound produced by the friction between a car’s tires and the road. We develop classifiers to recognize and localize cars approaching the user in real-time.

• We create, PAWS, a low-cost, end-to-end wearable system using COTS components accompanied with a smartphone application to provide real-time alerts of oncoming cars to pedestrians in noisy urban environments. We demonstrate that inattentive pedestrians can immediately benefit from our system.

• We present a second system, PAWS Low-Energy, that improves upon the power consumption of our COTS implementation by offloading critical and computationally expensive features onto an application-specific integrated circuit, developed by our collaborators [6]. We
additionally introduce **Angle via Polygonal Regression** (AvPR), an easily calibrated method for estimating the direction of arrival of car sounds. AvPR improves upon the granularity of direction estimates over the classification approach employed in PAWS and accommodates for noise better than classical geometric approaches (e.g. triangulation) for estimating direction, while remaining computationally inexpensive.

• We develop a segmented architecture and data processing pipeline that intelligently partitions tasks across the front-end hardware and the smartphone and ensures accuracy while minimizing latency.

As the industry is investing heavily in intelligent headphones [18][19], our hardware-software co-design approach presents a compelling solution towards protecting distracted pedestrians in urban environments.

### 2.1 Related Work

Object recognition and localization have been vastly explored in the literature. Almost all of them mirror techniques that are present in nature, such as the use of stereo imaging [20], ultrasonic radars [21], and acoustic source localization [22]. In vehicular tracking, video based approaches have been widely used [23][20][24]. The amount of information that can be extracted from images is undoubtedly greater than any other types of sensors. Commonality in vehicles’ shapes and standardized road signs have enabled machine learning algorithms to identify and predict the movement of cars [25]. Although such systems offer outstanding solutions for devices that can be hosted in large platforms, e.g. in an autonomous car for collision prevention [26], these are not suitable for use in wearable systems. A major limitation is the high computational requirements for real-time image processing. Another major issue is the privacy of the user. Video can reveal an alarming amount of personal identifiable information.

Active techniques like radar and LIDAR can certainly be used to detect the presence of obstacles and even some of its spatial behaviors [27][28], but such solutions face great challenges
in classifying what those obstacles are. This is particularly problematic in urban environments where moving and stationary obstacles are abundant, but only a few are real threats to the user. On the implementation side, the inherently high power dissipation of active transducers are usually discouraging for portable devices.

Passive audio sensors, on the other hand, provide enough information to allow classification and localization of the source with less computational and power requirements. But, unlike other techniques already published [29][22][30], PAWS uses machine learning algorithms to improve its predictions. By doing so, the system requires a large amount of learning data, but gains in flexibility, speed, and complexity. Audio classification has been used for event detection like, coughing detection [31], gun shot detection [32], human activity (e.g. talking, crying, running etc) detection [33]. These works mostly focus on identifying prominent sounds like gun shots or shouting rather than noise-like car sounds. [34] classified subtle sounds like keyboard typing, door knock etc. but all of the sounds were in an isolated environment not in real life noisy environment. [35] considered bus and trucks as events but had a very low accuracy of 24%. Other signals like video [36] [37] or seismic signals [38] have been used for vehicle detection but are not suitable for a wearable system like PAWS.

Other works that leverage sensors to enhance pedestrian safety utilize sensors placed on the shoe [39] or the camera on the smartphone [40], but these approaches are either unable to detect and localize cars or provide limited coverage.

In recent years, developments in vehicle to vehicle, pedestrian, and infrastructure communications are beginning to allow cars and pedestrians to directly communicate with each other in close proximity. Dedicated Short Range Communication (DSRC), a wireless protocol developed specifically for vehicular networking, is becoming a popular protocol in vehicular networks to transmit information and alerts [41][42], but is not natively supported on smartphones and require modifications to existing vehicles. Solutions that leverage standard WiFi or cellular protocols generally require every car to have a wireless transmitter [43][44], do not meet timing and latency requirements [45][46], or modify the SSID of smartphone WiFi beacons [47] and cannot be implemented
on a smartphone with standard privileges. Additionally, a single pedestrian only requires our cus-
tom headset to receive the full functionality of PAWS and PAWS Low-Energy. In contrast, V2X
systems commonly require modifications to support specific wireless protocols on all passing cars
before a single pedestrian can benefit.

2.2 Studying the Problem

Before developing PAWS into a wearable system, we studied the car sound recognition and
localization problem using a validation platform. The objective of this exercise was to analyze the
feasibility and complexity of our proposed solution and to determine the specifications required to
capture the necessary information, e.g., audio sampling rate, sensor placement, and most relevant
features to use in the machine learning algorithms.

As shown in Figure 2.2, the platform directly connects eight MEMS microphones to a com-
puter. The microphones were placed on a mannequin head to reproduce the physical phenomena
of the final setup, such as the acoustic shadow of the human head [48] and the approximate spacing
among sensors on a real user.

The study has been done in five different locations in two different cites: a metropolitan area
and a college town. The locations were two parking spaces, a four-way intersection, and two multi-
lane streets. We analyzed recorded audio from 47 different cars. Other than the parking spaces,
where we conducted our first set of controlled experiments with labeled distances, directions, and
precise time-keeping of honks and car passing, all other scenarios were uncontrolled.

2.2.1 Recording Specifications

In order to characterize the sounds of interest, such as an approaching vehicle’s tire friction, en-
gine noise, and honks, we conducted controlled experiments in two parking lots (Figure 2.2 shows
one of the experiments). These results are later cross-checked against uncontrolled experiments’
for consistency.

Figure 2.3 shows the spectrogram of one of the recordings from the controlled experiments.
Both the top and the bottom figures correspond to the same recording. Approximately 5s after the recording starts, a car honks, resulting in distinct stationary tones with fundamental frequencies near 500Hz. The vehicle then accelerates towards the mannequin. In the bottom figure, where the lower part of the spectrogram is highlighted, we see the engine noise. The engine noise follows its RPM. In an automatic car, the engine noise is bounded between 60Hz and 200Hz (at the 7 seconds mark, the shift in the engine gear is noticeable). Once the vehicle gets closer to the mannequin, the friction noise from the tires and asphalt gets louder. This noise has a band-limited spectrum with more energy below 3kHz. When the car crosses the system near the 12s mark, a burst of air causes a loud white noise. Similar spectrum components were found on several recordings of different cars at similar speeds (20-30mph) on dry asphalt.

These observations indicate that to identify warning honks and vehicles that are still approaching the user, the system audio must reliably capture frequencies from 50Hz to 6kHz. This requirement means that the system needs custom microphone drivers with a cut-off frequency of less than 10Hz (in contrast to standard headset microphones with approximately 100Hz cut-off frequency) and analog-to-digital converters with sampling rates above 12kSamples/s.
2.2.2 Presence of a Car

The presence of a car can be determined from high-energy, sharp sounds like honks, as well as from low-energy, noise-like sounds such as the sound of friction between a tire and the road. Being able to detect cars based on friction noise is crucial given the increasing popularity of electric cars with quieter engines.

Honks are louder and, thus, easier to detect than car tire or engine sounds. We analyze the Mel-Frequency Cepstral Coefficients (MFCC) [49] of honks and compare them with non-honk street sounds. We start with MFCC, since it is one of the most commonly used acoustic features for detecting various types of sounds [50][51][52][53] including car sounds [54]. For visualization
Figure 2.4: Distribution of honks and other types of sounds in a 2D feature space.

Purpose, we reduce the 13-dimension MFCC features to two dimensions (using PCA [55]) and the result is shown in Figure 2.4. We observe that honks are separable from other sounds as they cluster around a different point in space. Honks are easily detectable using all 13 coefficients.

MFCCs, however, are not effective in detecting other types of car noises such as friction between tires and the road. The fundamental reason behind this is that the Mel-scale, expressed by $m = 2595 \log_{10}(1 + f/1000)$, was originally designed to mimic human hearing of speech signals that maps frequencies $f < 1$kHz somewhat linearly, and maps $f > 1$kHz logarithmically. Our analysis on tire friction sounds shows that about 60% signal energy is attributed to frequency components below 1 kHz. Hence, to model such low-energy, low-frequency, noise-like sounds, we need to develop a new feature that captures these sub-kHz characteristics. We propose this new feature in Section 2.3.3.
2.2.3 Direction of a Car

To determine the direction, we record audio of cars approaching from different directions and analyze their effect on the microphone set. Some of these recordings also have honks in them. Intuitively, microphones that are closer to the sound source and are not obstructed by the human head should receive signals earlier, and the signals should be stronger. Hence, the relative delays and the relative energy of the received signals should be strong indicators of the direction of an approaching car.

In Figure 2.5, we plot the relative delays of the microphones with respect to the front microphone for left and right side honks. We see that the relative delays change signs for left and right honks. We do similar tests with eight directions (each covering a 22.5° 3D cone surrounding the mannequin) to successfully determine the directions of honks near the user.

Similarly, we plot the relative delays of the microphones for a car that passes the mannequin from its left to the right (Figure 2.6). We observe that the relative delays are quite random on both left and right ends. As the car approaches the mannequin, we see a trend in all the curves with one or more of them reaching their peaks. The trend reverses as the car passes the mannequin. This behavior suggest that patterns in relative delays (when they are looked at together) are useful to determine the direction of passing. Hence, by learning the trend and the point when the trend
reverses, it is possible to differentiate between a car on the left from a car on the right, as well as their angular directions.

2.2.4 Distance of a Car

In an attempt to estimate the distance, we formulate a regression problem that maps sound energy to distances. Later we realize that due to environmental noise and the weakness of car sounds, a fine grained location estimation is extremely inaccurate when the car is farther than 30 m from the audio recorder. When the car is within 30 m, we find that the maximum value of the cepstral coefficients (computed every 100 ms) is approximately linearly correlated with distance, as shown in Figure 2.7 for a car that is driven toward the mannequin. This relationship can be exploited to form a regression problem that maps maximum cepstral coefficients to distances.

For cars farther than 30 m, although we are able to detect their presence and estimate their direction, a precise distance estimation results in a large error. However, we learn that the distance estimation problem can be formulated as a multi-class classification task by dividing the absolute distances into a number of ranges such as (0, 20 m], (20 m, 40 m], and (40 m, 60 m]. Each of these ranges can be characterized with additional features, such as zero-crossing rate, and can be classified accurately using a machine learning classifier.

As such, one option is to use a two-level approach for distance estimation. The first level em-
Figure 2.7: The maximum cepstral coefficient follows a trend when an approaching car is within about 30m from an observer.

employs a classifier to determine a coarse-grained distance range, and if a car is detected within the nearest range, it applies regression to obtain a fine-grained distance estimate. A second option is to disregard estimating coarse-grained distance and only provide fine-grained distance measurements via regression. As shown in Figure 2.7, for distances above 30 meters, the maximal cepstral coefficient maps to approximately the same value. As such, the regression model inherently classifies car distances into two coarse groups: greater than 30 meters and less than 30 meters away. We considered both methods over the course of designing both phases of PAWS, as detailed in Sections 2.3 and 2.4.

2.3 Overview of PAWS

PAWS is a wearable headset platform and smartphone application that uses five microphones and a set of machine learning classifiers to detect, identify, and localize approaching cars in real-time and alerts the user using audio/visual feedback on his smartphone.

The system consists of three main components: sensors and their drivers, front-end hardware for multi-channel audio feature extraction, and a smartphone host for machine-learning based vehicle detection and localization, which are shown in Figure 2.8. Four of the MEMS microphones,
Figure 2.8: A block diagram of PAWS. The components highlighted in red are portions of the system that are enhanced and modified during the second phase of development to create PAWS Low-Energy, which is discussed in Section 2.4.

labeled MIC1 to MIC4, are distributed over the user, at the left and right ear, back of the head, and chest of the user, to provide relevant information about the sound source’s location. The front-end hardware synchronously acquires analog signals from these microphones and locally extracts acoustic features that are used by a smartphone application. PAWS performs signal processing inside the front-end hardware so that only features need to be transmitted wirelessly to the smartphone (via BLE) instead of large amounts of raw audio data. The front-end hardware is a battery-powered embedded platform that is housed within the confines of the headset that uses its own set of microphones for sound processing. It does not interact with the speakers or microphone of the headset. As such, a user would have not experience any degradation in sound or microphone quality of the headset.

The standard microphone of the headset (the fifth microphone, MIC5) is connected to the 3.5mm audio input of the phone. Data from the fifth microphone is directly acquired by the smartphone. The audio from the smartphone/headset microphone is acquired in the same way as com-
Figure 2.9: (Left) Teardown of the PAWS headset; the front-end hardware is exposed inside the left ear housing. (Right) Close up of the PCB that comprises the PAWS front-end hardware.

common messaging and calling applications and does not affect the quality or user experience of the microphone. Using the features computed by the front-end hardware and an audio stream from the headset microphone as inputs, machine learning classifiers running inside the PAWS application detects the presence of an approaching vehicle and estimates its position relative to the user. Our architecture uses a single low-power microcontroller in the front end and relies on the smartphone to run machine-learning classifiers to deliver reasonable latency.

2.3.1 Front-End Hardware

The front-end hardware is responsible for three blocks on the PAWS signal flow: synchronous ADC of microphone channels, embedded signal processing, and wireless communication with the smartphone. The integration of these blocks in a wearable resource-constrained system is a challenging task, and computational bottlenecks such as memory and data transfer rate require a careful distribution of resources.

In order to demonstrate PAWS’s system architecture and algorithms, off-the-shelf components were used to build the system. As shown in Figure 2.8, four MEMS microphones are wired to an MCU. The MCU synchronously collects the signals, calculates the temporal-spatial features,
and sends the result to a smart BLE module via UART. The BLE module sets the link between the front-end hardware and the smartphone. The front-end hardware is powered by standard AAA batteries and is designed to fit inside the left ear housing of a commercial headset, as shown in the left figure in Figure 2.9.

2.3.2 Front-End Signal Processing

In this section we discuss the operations that are processed by the front-end hardware. The MCU must sample the data from the four MEMS microphones and perform feature extraction, while the BLE module is responsible for transferring the calculated features to the smartphone. Since cars may be traveling at high speeds, fast response times and low latency are critical. PAWS uses a Cortex-M4 MCU to perform data acquisition and processing in real-time. The design choices and evaluation are explained in detail in Section 2.5.

Sampling Data

Audio is captured from four microphones at 32kSamples/s with an 8-bit successive approximation ADC and a four channel analog multiplexer running in the microcontroller. The sampling frequency was chosen as a compromise between the lowest rate necessary to capture the spectral content, as explained in Section 2.2, and the performance enhancement achieved by a delay estimation with finer granularity.

Feature Extraction

Running the feature extraction algorithms in real-time in a Cortex-M4 is challenging due to the complexity and number of computations required across the four channels. In order to service a continuous stream of incoming data, it is imperative that the feature extraction finishes before the next window of data is completely received. The feature extraction calculations were simplified to achieve low latency; complex multiplications and division were avoided. The following features were calculated on the acquired four channels of data: relative power of each channel with respect
to MIC1, relative delay with respect to MIC1, and zero-crossing rate of each channel. These
features are calculated for every time window of 100ms with 50% window overlap.

The relative power ($R_{P_{N,1}}$) is calculated by summing the difference of squares between samples
from each microphone to the reference microphone, MIC1.

$$R_{P_{N,1}} = \sum_{i=1}^{W_L} (X_N^2[i] - X_1^2[i]) \quad (2.1)$$

$N$ is the channel number, $W_L$ is the window length (in this case 3200 samples), $X_N$ is the
channel signal, and $X_1$ is the reference MIC1 signal.

The relative delay is calculated using cross-correlation. The lag between the channels is defined
as the index where the cross-correlation ($XCORR_{N,1}$) is maximum.

$$XCORR_{N,1}[d] = \sum_{i=0}^{W_L} X_N[i-d] \cdot X_1[i] \quad (2.2)$$

This is the most computationally expensive calculation of the front-end system. Since the phys-
ical separations of microphones are limited, e.g. the average spacing between ears is $\sim 25\text{cm}$, the
range of valid relative delay is bounded, making it possible to compute and compare the XCORR
only for $d \in [-40, 40]$. According to [56], these limits on the time interval of interest make com-
puting cross correlation in the time domain much more efficient than computing in the frequency
domain.

The zero-crossing rate ($ZC_N$) is the number of times a signal changes sign within a given time
window.

$$ZC_N = \sum_{i=1}^{W_L} (|\text{sgn}(X_N[i]) - \text{sgn}(X_N[i-1])|) \quad (2.3)$$
Figure 2.10: PAWS Smartphone data processing. The components highlighted in red are portions of the system that are improved upon in PAWS Low-Energy, which is discussed in Section 2.4.

**Data Transfer**

The BLE module gathers the resultant 10-element feature values and sends them to the smartphone following a custom protocol in 40 byte packets. The protocol consists of a validation header (3 bytes), followed by a set of hardware configuration flags (1 byte), payload size (1 byte), and the feature values ($1 \times 3$ bytes for relative delays of MIC $\{2, 3, 4\}$, $8 \times 3$ bytes for relative powers of MIC $\{2, 3, 4\}$, and $2 \times 4$ bytes for ZC of all four microphones).

### 2.3.3 Smartphone Data Processing

The PAWS smartphone app receives a 44.1kHz, single channel audio stream from the headset via the standard microphone jack, acoustic features over BLE from the front end, and processes them in real-time in a service. The application comes with a graphical user interface that is used to start/stop the service, configure alerts, and display a timeline of approaching cars along with their distances and directions.
Figure 2.10 shows the data processing pipeline of the PAWS smartphone application. The application implements a two-stage pipeline for detecting and localizing cars, respectively.

**Car Detection Stage**

Two offline-trained classifiers are used in this stage to detect cars honks and engine/tire sounds. The first classifier uses standard MFCC features to detect the presence of car honks. For the other type of car noises, we incorporate a new acoustic feature, *Non-Uniform Binned Integral Periodogram (NBIP)*, introduced by our collaborators from the University of North Carolina at Chapel Hill\(^1\), that unequally divides the frequency scale in order to capture variation in spectral energy at the lower end of the frequency spectrum which characterizes the friction sound from car noises. The steps to compute the NBIP features are as follows.

- **Step 1:** The FFT of each audio frame \(x(t)\) is computed to obtain the Fourier spectra \(X(f)\). Only the left half of this symmetric spectra is retained.

- **Step 2:** The periodogram of \(x(t)\) is obtained from \(X(f)\) by normalizing its magnitude

---

\(^1\)led by Shahriar Nirjon, Department of Computer Science, University of North Carolina at Chapel Hill: [https://www.cs.unc.edu/~nirjon/](https://www.cs.unc.edu/~nirjon/)
squared, and then taking its logarithm.

\[ P_x(f) = 20 \log_{10} \left( \frac{1}{F_sN} |X(f)|^2 \right) \]

\( F_s \) and \( N \) denote the sampling frequency and the signal length, respectively.

- **Step 3:** The frequency range is divided into a total of \( B \) bins, such that the frequencies below a threshold \( a \) are equally divided into \( b \) bins, and the higher frequencies are equally divided into \( B - b \) bins. The binning process is illustrated in Figure 2.11. The optimal values of the parameters \( B, a, \) and \( b \) are empirically determined, which we will describe shortly.

- **Step 4:** The \( P_x(f) \) is integrated in each bin to obtain a \( B \) dimension feature vector \( v = (v_1, v_2, \ldots, v_B) \).

\[ v_k = \begin{cases} \int_{(k-1)\Delta_1}^{k\Delta_1} P_x(f) df, & \text{if } 1 \leq k \leq b \\ \int_{a+(k-b)\Delta_2}^{a+(k-b-1)\Delta_2} P_x(f) df, & \text{otherwise} \end{cases} \]

where, \( \Delta_1 = \frac{a}{b} \) and \( \Delta_2 = \frac{1-a}{B-b} \) are the bin sizes for frequencies below and above the threshold \( a \), respectively.

In order to find the optimum values of the parameters \( a \), and \( b \), we vary the parameters \( 0 \leq a \leq 1 \) and \( 1 \leq b \leq B \) in small increments and compute the vector difference between features of car noises and all other non-car sounds. We observe that when \( a = 0.3 \) and \( b = 18 \), the vector difference between the car noise features and the non-car sound features is maximized. Figure 2.12 shows the mean and standard deviation of each component of the two types of feature vectors (i.e. NBIP and MFCC), for the two classes of sounds. We observe that most of the NBIP feature components (e.g., the first 10 components) are very dissimilar for the two classes, whereas the MFCC features for both classes are very similar. Unlike MFCCs, NBIPs are designed to maximize their vector representations for car engine/tire vs. non-car sounds, which makes them effective in recognizing cars with a very high accuracy.
The NBIP features introduced are only used to detect approaching cars/engine and tire noises. Since honks exhibit strong frequencies in narrow bands and are not noise-like, we cannot use NBIP to accurately detect honks. As such, we use standard MFCC features for honk detection. For both types of classification (honks vs. engine/tire noises), we train separate Random Forest classifiers \cite{57} which perform significantly better than other classifiers (e.g., Support Vector Machine \cite{58}) that we applied on our data set.

The classifiers were trained using audio recorded from 60 different vehicles, ranging from sedans to buses and trucks, including the initial 47 recorded for studying the problem. We found
similar performance using classifiers trained with sounds recorded from as low as 30 different cars. Additionally, we included environment sounds without cars recorded from the college town and metropolitan areas in which we conducted our study. These audio clips include a wide range of non-car sounds typically found in an outdoor environment, including but not limited to talking, wind, and wildlife.

**Car Localization Stage**

If the presence of a car is detected, the second stage of the pipeline is executed. In this stage, the smartphone acquires and uses the four-channel acoustic features received from the embedded front-end system to estimate the distance and direction of the car. Four multi-class Random Forest classifiers are used to classify eight directions and three distance levels based on honks and engine/tire-friction sounds, respectively. Because the feature vectors are only of 10 dimensions, we feed all the features into both classifiers for a simpler implementation. However, our analysis of principal components (PCA) reveals that relative delay and relative powers are more relevant features for direction classification, whereas relative delay combined with ZC and relative power are relevant features for distance estimation. Relative delay is relevant to the direction of the sound source because the microphone closer to the sound source will receive the audio signal sooner than the other microphones.

In addition to determining one of the three levels of distances, when a car is detected within the nearest level (within 30m), PAWS runs a linear regression-based fine-grained distance estimator. This step includes computing the cepstral coefficients and then fitting the maximum value to an actual distance in meters. This step does not add any significant cost as we obtain the cepstral coefficients as a byproduct of MFCC computation during the car detection stage.

**Alert Mechanism**

The application alerts a user with audio/visual feedback. If a car is detected within a user-configured distance range (e.g., 40m) – the phone vibrates, lowers the volume, and beeps. It can
also be configured to play a customized message, e.g., “a car is \{approaching, honking\} on your \{direction, left, right\}". The application also visually shows the location and direction of the car on its user interface, as shown in Figure 2.1.

2.4 PAWS Low-Energy

The PAWS front-end platform uses an MCU to sample and compute audio features used in direction and distance classification. Some of these features, such as cross-correlation, are computationally expensive and grow quadratically in window size. While the MCU-based sensing system we developed for PAWS is already optimized for power consumption, it still consumes significant energy since MCUs are general-purpose processing units that are not optimized for our specific application. To address this challenge, we further reduce power consumption of the front-end platform by incorporating an application-specific integrated circuit (ASIC effort led by Daniel de Godoy [6]) to compute digital acoustic features directly from analog sound sources. In this section, we introduce PAWS Low-Energy, an improved version of PAWS, that uses our custom ASIC in place of an MCU to sample audio and compute critical audio features for localization to further reduce power consumption. Additionally, we discuss the limitations of using the ASIC and how our localization methods change as a result. Finally, we introduce our computationally efficient, noise-resilient, mapping-based method for direction localization, Angle via Polygonal Regression (AvPR), that provides finer granularity direction estimates than the eight directions provided by the classifier-based method implemented in PAWS.

2.4.1 Reducing Power through a Custom Application-Specific Integrated Circuit

To reduce the power consumption of acoustic feature computation, we replace the MCU in the front-end platform with a custom-designed ASIC. We describe the design, fabrication, and evaluation of our ASIC in detail in [6]. This ASIC computes the relative delay between three channels of audio with respect to one channel of audio in the analog domain, using a technique called polarity-coincidence correlation. The polarity-coincidence correlation (PCC) between two
signals, \( X_1 \) and \( X_2 \), is shown in Equation 2.4 [59].

\[
PCC_{X_1,X_2}(\tau) = \int_{t}^{t+\tau} \text{sgn}(X_1(t)) \cdot \text{sgn}(X_2(t - \tau)) \, dt
\]  

(2.4)

The main difference between standard cross-correlation and PCC is that PCC computes the relative delay between two signals using only the signs of signals instead of the entire signals. Because only the signs of the signals are required, our custom ASIC does not need to sample entire audio streams to compute relative delay and is able to efficiently extract the relative delay through a feedback circuit using only comparators for computing signs and memory elements for shifting signals. This design leads to a reduction in power consumption of the front-end to nW levels compared to the mW level consumption by using ADCs on an MCU to directly sample the audio streams. Using the custom IC, we show in Section 2.5.2 that the majority of consumption is due to other components of the front-end, such as the BLE module. This reduces the overall power consumption of the wearable system by an order of magnitude, allowing us to power PAWS Low-Energy for longer than PAWS even after replacing the AAA batteries with CR2032 coin cells, which have one order of magnitude less energy capacity.

2.4.2 Resolving Constraints Imposed by Utilizing the ASIC

As mentioned in Section 2.3.2, PAWS uses a variety of features, including zero-crossing rate and relative power, for direction and distance classification. However, the ASIC only provides a relative delay measurement; thus, PAWS Low-Energy cannot utilize the same set of methods used in PAWS.

The distance classifier introduced in Section 2.3.3 uses relative power and zero-crossing rate, two features not computed by the ASIC, to classify the distance of a detected car into three coarse ranges. If the classifier detects that a car is within 30 meters of the user, we fit the maximal cepstral coefficient computed from the audio stream sampled from the phone to a distance learned
via regression. Past 30 meters, the maximal cepstral coefficient levels off to a base value as shown in Figure 2.7, and we cannot distinguish different distances past this point. This model inherently provides coarse distance classification: greater than 30 meters away and less than 30 meters away. Additionally, if a car is further than 30 meters away from a pedestrian, it is not critical for the user to know the exact distance of the car. As such, PAWS Low-Energy entirely removes the coarse-grained distance classifier and uses only the regression model introduced in Section 2.2.4 for both coarse and fine-grained distance estimation.

In the direction classifiers employed in PAWS, relative delay and relative power features are used for classification. Since relative power is no longer available, we can only use relative delay to compute the angle of arrival of the car. According to [60, 61, 30], relative delay is sufficient for computing the direction of arrival of a sound source. One option is to train another eight direction classifier using only relative delay features. If the position of the microphones are known in advance, a second option is to employ classical geometric methods, such as triangulation, to directly compute the location of the source with respect to the user. However, the accuracy of triangulation methods depends heavily on the distance between microphone sensors and the sampling frequency of the audio streams.

Despite the higher localization granularity that can be achieved, a small amount of noise can cause large amounts of direction and distance error in classical geometric methods. In the following section, we present AvPR, a training-based direction of arrival estimation method that provides the same localization granularity as classical triangulation methods, while also being more robust to environmental noise.

2.4.3 Angle via Polygonal Regression

AvPR takes advantage of the idea of using data to build a model to learn a physical phenomena and reduce the influence of noise, similar to a machine learning classifier. In the PAWS system, we trained the direction classifier by having someone (or a mannequin) wear the headset and playing a sound source (either generated or a real car sound) in each of the eight directions we classify.
Figure 2.13 displays the relative delay measurements obtained by playing white noise in each of the eight directions. The coordinates of each three-dimensional point corresponds to one of the three relative delay measurements computed in the front-end. We see that samples obtained from all eight directions form a circular shape on a relatively flat plane in three-dimensional space. From this representation, we can easily see that each point within this circular structure maps to a direction with respect to the user. A standard machine learning classifier would disregard this insight and learn boundaries between each of these eight directions to classify future observations into one of these eight categories.

Instead, AvPR provides direction estimates with similar granularity to triangulation methods by creating a mapping between direction, $\theta$, and the circular structure that the relative delays exhibit. For each direction, $\theta_i$, that we take measurements from to build the AvPR model, there is a sample mean, $\mu_i$, and variance, $\sigma^2_i$. As shown in Figure 2.14, we can approximate the structure of the relative delays, and therefore any angle between two angles for which we have observations for, as an $n$-sided polygon by interpolating between adjacent direction observation means and variances using linear splines, where $n$ is the number of directions we use to calibrate the model (eight is shown in the example). Now we have interpolated a probability distribution, $x_\theta \sim D(\mu_{x\theta}, \sigma^2_{x\theta})$ for all possible directions, $0 \leq \theta \leq 2\pi$, where $x$ is the vector of relative delay observations obtained from the ASIC.

Now that the model is built, whenever a new observation of relative delays, $x$, arrive, we can estimate the direction, $\theta^*$, by optimizing over the observed and interpolated direction distributions. There are multiple ways to accomplish this, but a common method is to use a maximum likelihood estimator and assume that a car has an equal probability of appearing at any direction, as shown in Equation 2.5.

$$\theta^* = \max_{0 \leq \theta \leq 2\pi} P(x|\theta) \quad (2.5)$$
This optimization problem is computationally expensive because we have to optimize over an uncountably infinite number of angles. We considered two ways to simplify this optimization. The first method is to quantize the angles into bins and optimize over the finite number of bins. However, this method is very similar to classification: the less bins we divide the angles into, the lower the computation and granularity of our estimator. The second method is to simplify the model by assuming that the variance, $\sigma^2_{\theta}$, of all directions $\theta$, are equal. Then, the optimal angle corresponds to the point on the polygon that the observation is closest to, which only involves computing the length of the normal segment from the observation to each side of the polygon, as shown in Figure 2.15. If the normal segment extends beyond the polygon, then the distance between observation and the polygon side is computed as the distance between the observation and the closest endpoint of the polygon segment. We adopt the second optimization method into AvPR because we preserve the granularity of direction estimation, while also being computationally efficient.

In addition to being more granular than the classification method, AvPR also requires less calibration points for greater granularity than the classifier method employed in PAWS. In theory AvPR has the same granularity as the classical triangulation methods as long as the clusters of relative delay calibration points can form a simple polygon. This is because a simple polygon is two-dimensional and can capture all angles 360 degrees around the user. Since the simple polygon with the least number of vertices is a triangle, AvPR can obtain the same granularity of direction estimates as triangulation methods while only requiring three training directions to build the model. The classification methods require $n$ calibration directions to categorize new observations into $n$ different directions. In Section 2.6, we compare AvPR with the classifiers employed in PAWS as well as an implementation using triangulation and show the robustness of AvPR over these existing methods. We summarize the steps for building the AvPR model and estimating new directions.

1. **AvPR Training Phase**

   (a) Play sounds at $n$ directions around headset and record the relative delays as features to obtain calibration directions.
(b) Interpolate between the sample means of adjacent directions. The resulting $n$-sided polygon in the feature space corresponds to the car’s angle of arrival around the user.

2. **Estimating Direction with AvPR**

(a) For a new observation, $x$, containing the relative delays sampled from the front-end platform, find the point on the trained polygonal model that is closest to the new observation in the feature space of relative delays.

(b) This closest point maps to the direction that AvPR estimates the sound source is coming from.
2.4.4 PAWS Low-Energy System Architecture

The PAWS Low-Energy front-end and smartphone block diagram has the same flow as PAWS, but some of the modules within the flow are modified and updated using techniques introduced in this section. In the front-end platform, PAWS Low-Energy replaces the MCU with the ASIC. This switch occurs in the module enclosed by the upper red box in Figure 2.8. In the smartphone pipeline, the phone no longer receives relative power and zero-crossing rate features. The lower left red box in Figure 2.10 marks the area of this difference between PAWS and PAWS Low-Energy. Additionally, the direction and distance classifiers for localization in PAWS are replaced by AvPR and regression of maximal cepstral coefficients respectively. The lower red box in Figure 2.8 along with the upper and lower right boxes in Figure 2.10 show where these differences occur in the front-end and smartphone data pipelines. The car and honk detectors remain the same between...
Figure 2.15: Estimating a new direction with AvPR. Blue dashed lines are example cases of computing the shortest distance from the new point to a segment. The solid black line denotes the path from the new point to the point on the polygon that is closest to the new point.

both PAWS and PAWS Low-Energy.

2.5 Platform Evaluation

In this section, we first analyze and compare the real-time performance and timing analysis of each component in the PAWS and PAWS Low-Energy systems. Second, we analyze and compare the power consumption of both systems.

2.5.1 Real-Time Performance

In this section, we discuss and compare the real-time performance of PAWS and PAWS Low-Energy, the timing constraints involved, and the design decisions involved to meet them. Specifically, we will analyze the timing on the feature extraction in the front-end platform (the headset),
the BLE transmission from the front-end to smartphone, and the detection and localization pipeline in the smartphone. Response time is crucial for our system, as a few milliseconds can make a difference in saving the life of a user.

**Front-end Feature Extraction**

The first part of the data flow in PAWS and PAWS Low-Energy is sampling audio and extracting features. In PAWS, the embedded front-end hardware is handling 32kSamples/s with 8 bits per sample for each of the 4 channels with MEMS microphones. To minimize latency, we compute features in 100ms windows every 50ms in a pipeline fashion. This means that features are being calculated every 50ms with 50% window overlap. The MCU uses a dedicated ADC module with direct memory access (DMA) to leave more CPU cycles available for feature calculation. The ADC is continuously sampling audio and storing them in RAM while features from the previous frame are being calculated. The data transfer from the MCU to the BLE module is also done via a dedicated UART module. In order for this pipeline to work in real-time, all features from the current frame must be calculated before the acquisition of the following frame ends, and the UART module must finish sending the current feature vector before the next feature is ready to be sent. The timing of the different parts of this pipeline can be seen in the upper flow of Figure 2.16. The features calculation consumes 36ms of the available 50ms in one time slot, and the UART module completes each feature vector transmission in 1.9ms.

In PAWS Low-Energy, the MCU is replaced with the ASIC. The ASIC extracts the sign of the audio stream and provides 8-bit relative delay measurements at 50kS/s per channel for three channels. However, we are unable to transmit 50kS/s per channel worth of 8-bit relative delay measurements to the smartphone due to bandwidth limitations of BLE. To keep the same wireless transmission rate of PAWS, we only read one set of relative delays per channel every 2500 samples and transmit to the phone. This yields a transmission rate to the phone of 20 measurements per second. In order for the BLE module to support a transmission rate of 20 measurements per second per channel, the module must be able to read one set of relative delay measurements in less than
Figure 2.16: Pipeline of the PAWS and PAWS Low-Energy feature extraction process. In the PAWS flow, the “Features Calc.” block represents all the operations involved in the features extraction, and “TX” represents the UART communication between the MCU and BLE module. In the PAWS Low-Energy flow, “Extract” refers to the BLE module reading relative delays from the ASIC. 50 ms from the ASIC, which uses a custom communication protocol. From our measurements, the BLE module requires 9μs to completely read one relative delay measurement from each channel, which fits our timing requirements for real-time operation. The timing for the ASIC to BLE module extraction for PAWS Low-Energy is shown in the bottom flow of Figure 2.16.

Since the BLE module is directly reading the relative delays from the ASIC, it does not need to spend time to compute relative delay; it can directly send this value to the smartphone. As such, PAWS Low-Energy directly sends data to the smartphone after only spending 9μs extracting a measurement from the ASIC, while PAWS requires 1.9ms for the BLE module to read the relative delay measurement from the MCU on top of the 36ms of computing the relative delay. This is a significant decrease in latency over PAWS, which will effect the wireless transmission latency test, detailed in the following section.

**Front-end to Smartphone Wireless Transmission**

Another crucial timing aspect of the system is the latency to transmit the features from the BLE module to the smartphone. This latency will not only add to the response time of the system, but
it can also cause a mismatch between the vehicle detection and its localization. If the temporal-spatial features calculated in the front-end hardware take too long to reach the smartphone, the location estimation displayed to the user might refer to a different sound source than the vehicle that the system just detected. To verify that the smartphone will receive the data within an acceptable time interval, an adaptation to the system was made, as shown in Figure 2.17. A button was simultaneously connected to one of the inputs of the front-end hardware and the microphone input of the smartphone (as the regular microphone buttons). A verification app was developed to compare the difference between the time when the button-press event was detected by the smartphone application and when the smartphone received the data packet containing the same event. All aspects of the systems, including firmware, remain equivalent to the setup for standard operation. Both PAWS and PAWS Low-Energy were tested.

The average delay of the PAWS wireless transmission latency is on the order of 55ms as shown in Figure 2.18. Since the event can be captured by the MCU anywhere within the 50ms sampling windows, this latency is not expected to be lower than the 38ms required for the calculations and transmission. However, due to randomness in the delay on the smartphone path, a few samples on the histogram have lower latencies. Figure 2.19 shows that the wireless transmission latency of PAWS Low-Energy is around 27 ms, which is much lower than PAWS. This huge improvement is from the addition of the ASIC, as explained in the previous section. The ASIC updates relative delay measurements at every audio sample, and the BLE module must only read from the ASIC. However in PAWS, the MCU must wait for an entire window (50ms) of samples before spending more than half of a window (36ms) extracting and transmitting features to the BLE module, which adds significant delay to the pipeline. As such, we have shown that PAWS Low-Energy improves upon the front-end feature extraction + wireless transmission latency of PAWS.

**Smartphone Processing**

Figure 2.20 shows the execution times of various components inside the smartphone application of PAWS and PAWS Low-Energy. The application runs four threads in parallel. Thread 1 is
Figure 2.17: Block diagram of the test setup for the latency between the features from the front-end hardware and the smartphone.

Figure 2.18: Histogram of PAWS front-end hardware to smartphone latency acquired with the test setup shown in Figure 2.17.

responsible for getting audio data using the single channel microphone for car detection. We have taken 10 frames per window (448ms) for robust feature calculations. Thread 2 is responsible for receiving acoustic features over BLE. Thread 3 runs the car detector, which takes 86ms. In addition
Figure 2.19: Histogram of PAWS Low-Energy front-end hardware to smartphone latency acquired with the test setup shown in Figure 2.17. We see that the mean latency is shorter than PAWS.

Figure 2.20: Execution times of various components of the PAWS and PAWS Low-Energy smartphone app.

to car detection, thread 3 also runs the distance and direction estimators. PAWS requires merely 2 ms to classify distance and direction because these classifiers use precomputed features from the headset. PAWS Low-Energy requires a slightly less, though similar, amount of time to run its localization algorithms. The UI thread (Thread 4) takes 3ms to update the UI and to notify the user.
Table 2.1: Power Consumption and Price Breakdown of PAWS and PAWS Low-Energy (PAWS Low-Energy Total in Bold)

<table>
<thead>
<tr>
<th>Component</th>
<th>Idle [mA]</th>
<th>Active [mA]</th>
<th>Unit Price [U$]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCU (STM32f4)/ASIC</td>
<td>4.37/~0</td>
<td>50/~0</td>
<td>3.20</td>
</tr>
<tr>
<td>BLE Transciever (nRF52)</td>
<td>0.46</td>
<td>7</td>
<td>6.40</td>
</tr>
<tr>
<td>MEMS Mics × 4</td>
<td>0.48 × 4</td>
<td>0.48 × 4</td>
<td>0.40 × 4</td>
</tr>
<tr>
<td>Amplifiers × 4</td>
<td>2.34 × 4</td>
<td>2.34 × 4</td>
<td>1.60 × 4</td>
</tr>
<tr>
<td>Regulators</td>
<td>0.1/0.6</td>
<td>0.1/0.6</td>
<td>0.50/3.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>16.21/11.84</td>
<td>68.4/18.9</td>
<td>18.10/20.60</td>
</tr>
</tbody>
</table>

for both systems as well. The worst case execution time for the PAWS and PAWS Low-Energy apps is 91ms. Because we use a 50% overlap between successive windows for car detection, the PAWS app runs the full classification pipeline every $\frac{448}{2} = 224$ ms, and detects and localizes cars in 91ms (i.e., in real-time), giving users plenty of time to respond to oncoming dangers.

2.5.2 Power Consumption and Price Breakdown

We evaluate the energy consumption of PAWS and PAWS Low-Energy by measuring the power consumptions for both the embedded platform and the smartphone during idle and active states. In the active state, data is processed, features are computed, and results are transmitted to provide danger feedback to the user, whereas in the idle state, the smartphone application is not connected to the headset and most of the clocks in the embedded front-end platform are turned off to conserve power. The sole purpose of the idle state is to conserve power when the user is not using the system (e.g. when the headset is not paired with the phone).

The PAWS embedded platform uses an STM32f4 Cortex-M4 chip as the MCU that samples and extracts features, as well as a BMD-300 module that acts as the BLE transceiver. Operating at 180MHz clock speed, the STM32 MCU consumes the most power at 50mA when active. While not in active use, the power can be reduced to 4.37mA. The Cortex-M4 architecture provides a familiar environment for firmware development with an acceptable energy footprint and a low cost of U$3.20 at major part suppliers. The BMD-300 BLE transceiver module transmitting at 0dBm power consumes 7mA when active and consumes 0.46mA when in idle mode, only transmitting
advertisement packets. The BMD-300 module integrates the Nordic nRF52 BLE chipset and antenna in a small footprint component that fits this application for a low price of U$6.40. The other components of the front-end hardware are the 3.3V regulator, the MEMS microphones, and the pre-amplifiers. They consume 0.1mA, 0.48mA, and 2.34mA per component, and cost U$0.50, U$0.40, and U$1.60 per unit respectively. The overall power consumption of the system is below 70mA, allowing for 17 hours of continuous operation when powered by 3 AAA Alkaline batteries.

The main source of power consumption in PAWS Low-Energy is the BLE module as the ASIC has nW-level consumption. The same MEMS microphones and pre-amplifiers from PAWS were used in PAWS Low-Energy. The ASIC requires multiple voltage inputs to function. As a result, multiple regulators are required, increasing power consumption and price. Since the ASIC requires almost no power to operate, the overall power consumption of the PAWS Low-Energy front-end is more than three times lower than PAWS, allowing for more than two days of continuous operation if powered by standard AAA Alkaline batteries and around half a day of continuous operation if powered by standard CR2032 coin cell batteries. Table 2.1 quantifies and compares the power consumption and price breakdown of PAWS and PAWS Low-Energy. The components of PAWS Low-Energy is only around U$2.00 more expensive than PAWS, but result in significant power savings. To obtain the cost estimate for PAWS Low-Energy, we made the assumption that the ASIC would cost the same amount as the STM32f4 MCU if mass produced.

For the smartphone, the most energy consuming component is the display, which is only used to configure the app, and therefore, it is not necessary to keep it always on. The BLE communication consumes about 0.2mA. The energy consumption for the rest of the application is between 0.3uAh to 0.8uAh per frame for both PAWS and PAWS Low-Energy.

2.6 Real-World Evaluation

To evaluate the end-to-end performance of the complete PAWS and PAWS Low-Energy systems in realistic settings, experiments were conducted in three environments: 1) a street inside a university campus and a residential area, containing pedestrian-borne sounds such as walking and
talking; 2) by the side of a highway, with wind being the most prevalent non-car sound; and 3) in a metropolitan area, where both pedestrian-borne and wind sounds are common.

2.6.1 Experimental Setup

1. **Campus street and residential neighborhood.** The first experiment was done in a campus street and a residential neighborhood with a speed limit of 25mph. The background sounds in this environment were mainly pedestrian-borne (e.g. groups of people walking and talking). To evaluate PAWS, we used three fixed markers (yellow cones) on the sidewalk and the PAWS app to evaluate the detection, direction, and distance accuracy. Every time a vehicle passed a cone, a volunteer raised a flag and the event was logged in the PAWS smartphone application. The setup is shown in Figure 2.21. The experiment was repeated multiple times. Each time the user faced the road at a different angle, $\theta$, so that we could test the accuracy of the direction and distance estimation for as many different angles as possible. For PAWS Low-Energy, we recorded time-stamped video to obtain the ground truth.

2. **Side of highway.** The second experiment was done by the side of a highway (NC HWY-54) where we observe a constant flow of cars of diverse models, e.g., sedans, SUVs, trucks, and buses. The speed limit for the vehicles in this segment of the highway is 45 mph as it is close to residential areas. In this experiment, we were exposed to less pedestrian-borne noise, but experienced heavy
Table 2.2: Summary of Deployment Events.

<table>
<thead>
<tr>
<th>Deployment</th>
<th>User (Facing Angle)</th>
<th>Honks</th>
<th>Car Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metro Area</td>
<td>$0^\circ$, $\pm45^\circ$, $180^\circ$</td>
<td>48</td>
<td>165</td>
</tr>
<tr>
<td>Campus</td>
<td>$0^\circ$, $\pm45^\circ$, $90^\circ$, $\pm135^\circ$, $180^\circ$</td>
<td>0</td>
<td>97</td>
</tr>
<tr>
<td>Highway</td>
<td>$0^\circ$, $\pm45^\circ$, $90^\circ$, $\pm135^\circ$, $180^\circ$</td>
<td>0</td>
<td>65</td>
</tr>
</tbody>
</table>

wind noise due to the location of the highway and because we were in hurricane season. For ground truth collection, we marked the road in the similar way as we did in the campus street. However, unlike the campus street, cars on the highway were driven at a higher speed (around 50-55 mph) and they were large in number. Therefore, instead of appointing human volunteers, we recorded a time-stamped video and analyzed the video offline to obtain the ground truth.

3. Metropolitan area. The third experiment was done in the streets of Manhattan, New York, where the number of cars, adjacent streets, and buildings in the surrounding area is very dense. This environment is replete with sounds commonly found in the first two scenarios, including wind and the bustle of pedestrians and wildlife (e.g. birds and pets). Just as with the second experiment conducted near a highway, a time-stamped video was recorded and analyzed offline to obtain ground truth. To evaluate the detection, direction, and distance classifiers of PAWS and PAWS Low-Energy, the estimator outputs were logged and compared against the ground truth. During all the experiments we simulated a distracted pedestrian’s ability to detect cars by logging car events while listening to music in parallel with PAWS.

Table 2.2 provides statistics of the deployment in all environments. The table shows how the PAWS and PAWS Low-Energy user faced the road, and the number of logged honks and car events. Though the experiments presented in this section were conducted with a stationary user, we observe similar performance when the user moves at walking pace. We did not run experiments in scenarios where the user is moving at higher speeds, but will explore this avenue in future work.
2.6.2 Results

Car Detection

We measure the car detection accuracy of PAWS and compare its performance with that of the ground truth collector’s and distracted user’s reports. Since PAWS Low-Energy uses the same car detector, the results of PAWS Low-Energy are aggregated with the results of PAWS. Figure 2.22 compares the exact counts of total logged approaching car events for all environments. We see that almost all the cars logged by the ground truth collector have been identified by PAWS, whereas the distracted participant missed about 19%-36% of them. This shows that PAWS is a highly efficient system for detecting and alerting pedestrians of approaching cars. In summary, the car event detection accuracy is 97.30%, 99.48% and 95.59% in metro area, campus and highway respectively. Additionally, a confusion matrix for the detection classifier running on PAWS is presented in Figure 2.23 for the metro area. The difference in the counts shown in Figure 2.23 and Figure 2.22 is that in Figure 2.22, the values correspond to car events. For instance if one car passes by the user, it will count as a single car event in this figure, but PAWS will have multiple frames or windows it processes that will show up as car detections from the raw classifier outputs. Figure 2.23 on the other hand displays the confusion matrix for each individual frame computed by the PAWS application. We see that only one frame was misclassified as a noncar in the case where a car was present, and around 5% of the noncar samples were misclassified as cars. These values show that PAWS has fairly low false positive and false negative rates as well as high true positive and true negative rates. In other words, PAWS is able to correctly detect the presence of cars and reject cases where no car is present in common urban environments, rich with wind and pedestrian-borne noises in the background.

Localization Performance

In this section we will present evaluation on the direction and distance estimators of PAWS and PAWS Low-Energy. First, we compare the performance of the direction estimators of PAWS, the
AvPR method employed in PAWS Low-Energy, and the classical triangulation method. The results of the PAWS direction classification is shown in Figure 2.26 for all environments and the eight
directions that we trained the classifier to discern. We assume that the accuracy of the directions reported by the ground truth collector is accurate. Each reported direction from the ground truth collector is mapped to the classification results of PAWS. We observe that the average accuracy of the direction classifier over all directions is 86.7%.

Figure 2.24 plots the mean and variance of the estimated vs. true angle of the AvPR method employed by PAWS Low-Energy. We see that the average error over all directions is around 11 degrees, which is less than the 12.5 degree sections that the classifier in PAWS outputs. This shows that AvPR is more accurate and granular than the classification approach employed by PAWS. We also show the mean and variance of the estimated vs. true angle of the car computed via triangulation in Figure 2.25 as a comparison. The average error is around 45 degrees, which is larger than the error of AvPR. The standard deviation of error for AvPR is around 10 degrees compared to 44 degrees for triangulation, which shows that AvPR provides more consistent predictions than triangulation. The mean and variation in errors show that AvPR outperforms AvPR for direction estimation.

Next, we present the results of the distance estimators of PAWS and PAWS Low-Energy. Re-
call that PAWS performs coarse distance classification and only performs fine-grained distance estimation if the car is detected to be within 30 meters, while PAWS Low-Energy leverages the coarse-grained estimator found inherently within the fine-grained estimator for coarse-grained and
fine-grained estimations. Figure 2.28 shows the combined confusion matrix for all distance predictions of the coarse-grained classifier. Each distance section presents an accuracy of 77.9%, 76.4%, and 72.3% for ranges from above 60m to 40m, to 20m from the user. The overall accuracy the coarse-grained estimator is 75.6%. Figure 2.29 plots the estimated distance vs. true distance of cars driving towards the user wearing our systems. We see that for cars within 30 meters, the distance estimator has an average error of 2.8 meters.

Figure 2.27 summarizes the direction and distance classifier results for all environments. We observe that the overall accuracy of the distance classifier is 63% – 78%, and that the average direction classifier ranges from 80% – 98.5% depending on the environment.

2.6.3 Limitations and Future Work

In this work, we present novel, audio-based, wearable methods for enhancing pedestrian safety that is effective in typical urban scenarios, providing accurate and real-time alerts of vehicle presence and location. However, we recognize that our system is still at the research level and not ready for commercialization, as there are several scenarios in which PAWS and PAWS Low-Energy are ineffective. We discuss these situations next.
Figure 2.28: confusion matrices for distance estimation.

Figure 2.29: Estimated distances for cars within 30m are on average 2.8m off of actual distances.

Noisy Streets

PAWS is designed to detect the presence of cars in real-world environments. Streets may contain diverse kinds of noise, some of which may be different from the ones we have calibrated
our system for. PAWS should be trained in as many scenarios as possible for robustness.

Additionally, just as if a camera in a vision-based approach is unable to see the car (e.g. car is not in line of site or if it is dark outside), if for any reason the microphones are unable to pick up the sounds of the car (e.g. microphones are covered or noise overpowers everything in the environment), then PAWS would be rendered ineffective. We are currently looking into other sensing modalities, vehicular networking methods, and crowdsourcing to overcome the limitations in our audio processing methods.

Nearby Cars

The current design of PAWS considers only the positions of vehicles relative to the user, but not their trajectories. We can foresee occasions where a pedestrian is walking parallel to a busy road, and the system is giving warnings, even though the user is not in danger of being hit. Future work that takes into account the trajectory of both the vehicle and the user is under development.

Multiple Approaching Cars

The presence of multiple cars approaching the user can impair the reliability of the system. It does not matter if there are multiple cars present or if there is only a single car, PAWS will reliably detect the presence of vehicles, using the methods presented in Section 2.3.3, at no additional computational cost if multiple cars are present. The localization portion of PAWS and PAWS Low-Energy both use relative delay computed via time-domain cross-correlation methods, which are dominated by the highest energy source (e.g. the loudest vehicle). In summary, PAWS can detect the presence of vehicles and localize the loudest vehicle. PAWS cannot estimate the number of vehicles present, nor localize multiple present vehicles. We are currently investigating sound source separation and multiple sound source localization techniques to overcome this challenge.

It should be noted that although PAWS is unable to localize multiple oncoming cars, the majority of accidents occur when there are less cars in the streets. According to the National Highway Traffic Safety Administration (NHTSA), 70% to 80% of all pedestrian-related accidents in the
United States occur between 6PM to 6AM [62], when there are arguably less cars on the road than during the day. Additionally, the NHTSA also reports that 90% of accidents that included the death of a pedestrian involved a single vehicle. These statistics suggest that despite its shortcomings in handling multiple car cases, PAWS can still handle the more common scenarios involving accidents with fewer vehicles.
Chapter 3: Adaptive Acoustic Noise Filtering for Improving Construction Worker Safety

Just like in pedestrian safety, vehicles are one of the largest cause of construction worker injury. In fact, motor vehicle crashes are the number one cause of work-related deaths in the United States [63, 64]. These accidents arise in part because the worker is distracted by their work. In this chapter, we present CSafe, a low-cost wearable and smartphone platform, that can be easily integrated into common wearables such as helmets, hats, and headphones. CSafe leverages an array of low-power microphones, signal processing, and machine learning classifiers to detect and localize oncoming vehicles, and alert construction workers in real-time.

In addition to the challenges of latency and limited computational resources, construction worker safety is more challenging than pedestrian safety because construction sites are very noisy. The power tools that construction workers will be operating by the side of the road can be orders of magnitude greater than the engine and tire sounds of an approaching vehicle, which will adversely affect any acoustic detection or localization algorithm we may decide to implement.

Much like in pedestrian safety, we chose to create a hardware platform consisting of an array of microphones sampled by a low-power embedded platform. The embedded platform performs feature extraction and transmits wirelessly to the more powerful smartphone platform, which runs signal processing and machine learning algorithms to perform vehicle detection and localization, and send alerts to the user. Our embedded platform can be easily integrated into common wearables, such as headphones, helmets, and hats. In this work, we chose to integrate our hardware platform into a construction worker helmet, as shown in Figure 3.1.

To account for loud construction tools that will be present in a construction site we propose an energy-efficient sound filtering architecture that contains content-based separation and spatial sep-
Figure 3.1: (Left) The CSafe embedded platform, consisting of an array of four microphones integrated into a feature extraction system introduced in [8]. Two microphones are shown (the front and right microphones), while the other two are on the back and left side of the helmet. (Right) Screenshot of CSafe’s smartphone system.

aration in a feedback configuration, which utilizes known or learned models of sounds to iteratively remove noise and boost target sounds. Our goal is to show that our novel filtering architecture can be used in conjunction with existing vehicle detectors to boost performance in noisy construction site environments rather than serve as a complete replacement. We first develop a novel noise filtering algorithm called Probabilistic Template Matching (PTM). PTM is a low computation source separation algorithm that leverages statistical "templates" of noises to filter out these sounds. Next, we develop a novel noise filtering architecture that intelligently leverages PTM and multi-channel filtering methods to robustly filter out construction tool sounds from the environment. Our novel filtering architecture differs from existing works in sound source separation in that we intelligently leverage both single-channel source separation and multi-channel source separation in a feedback architecture to more robustly remove overpowering construction noise over time. We show that our novel filtering architecture can run in real-time on a low-resource embedded + smartphone platform and can improve vehicle detection by up to 12% more than other state-of-art source separation algorithms.

We summarize the major points of this chapter next:
• We create, **CSafe**, a low-cost, end-to-end wearable system to provide real-time alerts of oncoming cars to construction workers even in environments with tools that are orders of magnitude louder than approaching vehicles. We perform real-world experiments and show an improvement in vehicle detection by 16% and a 30° reduced localization error over existing systems.

• We develop a novel and light-weight single-channel source separation algorithm, called Probabilistic Template Matching (PTM) that uses learned statistical models or "templates" of construction tool sounds to filter them out.

• We develop an adaptive and selective noise filtering architecture that allows users and applications to select specific types of construction sounds to filter out. Our architecture integrates both PTM and multi-channel source separation methods in an adaptive feedback architecture to robustly filter out overpowering construction tool noise over time.

• We show that by applying our novel adaptive filtering architecture to vehicle detection, we can improve the overall vehicle detection accuracy by up to 15%. We also show that our architecture improves the vehicle detection rate by up to 12% more than other state-of-art audio source separation methods.

### 3.1 Related Works

Pedestrian safety has similar aims as construction worker safety, in that both attempt to reduce the number of vehicle accidents. However, the technical challenges between these two problems are very different. In construction worker safety, the presence of construction tools, that are many orders of magnitude louder than that of approaching vehicles poses a challenge that is typically not seen in pedestrian safety. In these scenarios, the sounds of loud tools effectively mask the sounds of oncoming vehicles.

There are a variety of commercial products and research works that address construction worker safety, including jackets, helmets, and smartphone systems, that monitor fatigue and posture [65,
or provide cooling and heating relief [67]. These methods do not warn workers of oncoming dangers such as approaching vehicles. There are also works and products that deploy sensors (e.g. RFID or proximity sensors) on either the construction worker or large equipment. These sensors can then be used to quickly locate workers in case accidents occur [68] or can send an alert to the worker if s/he comes too close to dangerous equipment [69]. These works require an installation phase in the construction site and cannot account for passing non-construction vehicles.

3.2 Overview of Source Separation and Selective Noise Filtering

It is difficult to detect and localize vehicles using audio signals in real-time and on a resource-constrained platform. Our problem is made even more challenging because construction workers work in sites that are extremely noisy, with loud machinery and tools prevalent in the environment.

To clean and remove overpowering construction tool sounds from the environment, we consider incorporating audio source separation methods into CSafe. We recognize that there are decades of work on noise filtering and source separation and that there is no way for us to address all aspects on the topic. As such, we summarize these works into two broad groups: spatial separation and content-based separation, that we will discuss next.

3.2.1 Spatial Source Separation

Spatial source separation techniques rely on observing multiple observations of the environment with multiple microphones in placed in different locations. Because these methods require multiple microphones, these methods are also called multi-channel source separation methods. Some methods that fall into this category include beamforming [70, 71, 72], general adaptive filtering techniques (e.g. least means squares and weiner filtering) [73, 74], blind source separation techniques (e.g. independent component analysis) [75, 76]. With the exception of blind source separation techniques and adaptive filtering, spatial source separation techniques generally require the location of the source to perform separation. As such, many of these works assume the location of the source is known in advance, which is not true in a dynamic urban environment.
Traditional blind source separation techniques do not require the location of the source to be known in advance. However, these techniques generally perform poorly in real-world scenarios [77].

Adaptive filtering techniques, such as least means squares, filter out noise by observing a second correlated noise signal. In the context of audio wearables, we would require a two microphone setup where one microphone observes the environment for vehicles. In a construction setting, this microphone would also observe construction tools sounds. A second "noise" microphone would ideally observe just the noise of the construction tools, which would be used to clean the signal from the first microphone. However, the position of the microphones in a practical wearable would be close together. If we were to integrate microphones into a helmet, then both microphones would observe relatively similar signals (e.g. the "noise" microphone would also observe the sound of passing vehicles). The denoising process would not only reduce the sound of construction tools, but that of passing vehicles, adversely affecting vehicle detection.

In light of these shortcomings for blind source separation and adaptive filtering, we decide to incorporate ideas in beamforming into our filtering architecture (Section 3.3.2).

3.2.2 Content-Based Separation

The second class of source separation algorithms is content-based separation, which uses learned knowledge and statistical models about specific sounds and noises in the environment to filter them out. Since these methods require data, but not multiple channels of audio, these methods are also known as single-channel source separation methods. Classes of techniques that fall into this category include but are not limited to dictionary learning (e.g. non-negative matrix factorization or NMF) and deep neural network methods [78, 79, 80, 81].

Many deep learning architectures that perform source separation require millions of parameters and a large amount of training data. For example, the network presented in [80] requires almost 2 million parameters to separate out two sources from a single channel of audio sampled at 16 kHz. Second, source separation neural networks must be trained using artificial mixtures. This is
because neural networks require the ground truth signal to adapt weights for training; in our application, the ground truth signal would be the isolated vehicle sound. However, it is not possible to extract the isolated vehicle sound in a real-world mixture because the vehicle sound is corrupted by construction tool noise. Creating a network with these two requirements will not produce a robust and low-power solution for filtering out construction sounds and improving vehicle detection and localization. We present experiments to back this in Section 3.4.

Dictionary learning methods learn a set of bases or a "dictionary" that capture most of the important features of a sound type. When a new signal arrives another optimization is performed to discover the coefficients or weights of each basis or "word" in the "dictionary" that the observed signal is comprised of.

We observed while experimenting with dictionary learning methods, like NMF, that the separation quality can be very poor because dictionary learning seeks to precisely deconstruct an entire signal into a weighted sum of its learned "words". However, our learned "dictionary" may not contain a learned representation of all sounds present. For instance, if we learn a "dictionary" for construction sounds, and someone begins speaking, NMF would attempt to fit construction sound "words" into speech, which would yield poor results. This brings up an assumption made by many content-based separation works: there is a model available for all the types of sounds present in our observation. This assumption is not always true in a dynamic environment, and it is not feasible to have a model of every possible sound in the environment. These points motivate a need for a fast content-based separation algorithm that can separate and filter out noises for which we have models for, while leaving all other sources, which we may not have prior knowledge for, intact.

In light of these shortcomings for content-based methods, we develop a novel light-weight single-channel source separation algorithm called Probabilistic Template Matching (PTM) that leverages learned models of construction sounds to filter them out. We integrate PTM into a novel filtering architecture described in Section 3.3.3 that uses a noise detector to control the level of filtering that PTM provides based on how dominant the construction sound is in the environment. In this way, our architecture only requires models of the noise (construction tools) and does not
require knowledge of all other sounds in the environment (e.g. vehicles) as dictionary learning algorithms typically require. The algorithm is also similar to general adaptive filtering techniques, as discussed in Section 3.2.1, but does not require a second microphone in close proximity, that would likely observe and diminish vehicle sounds as well.

### 3.3 CSafe Filtering Architecture

To perform robust urban and construction noise separation, we propose a novel adaptive filtering architecture, shown in Figure 3.2, that leverages both content-based separation and spatial source separation techniques. Our architecture incorporates both a noise detection module and a multi-source localization module required for both content-based and spatial separation and uses a feedback loop to adaptively learn better filter coefficients over time. We also propose a novel and light-weight content-based separation technique called *Probabilistic Template Matching* (PTM) that allows users and applications to tune the amount of noise suppression that CSafe provides. Our full architecture allows users to inject their own recordings of nearby tool sounds for more robust noise filtering. In the following subsections, we first introduce our novel content-based separation algorithm, Probabilistic Template Matching. Next, we introduce how we integrate our novel content-based separation algorithm with our spatial separation module to create an adaptive...
filtering architecture that intelligently leverages both multi-channel and single-channel separation techniques to more robustly filter out construction sounds and improve vehicle detection and localization. All figures of waveforms, mixtures, and quantitative analysis presented throughout this paper were generated with real-world recordings of mixtures over the air rather than digitally mixing sources, which is commonly performed in many works that propose source separation algorithms. This is to ensure that our examples and methods are representative of real-world scenarios.

3.3.1 Content-Based Separation: Probabilistic Template Matching

We propose Probabilistic Template Matching (PTM) a light-weight content-based source separation algorithm to filter out construction noises, while reducing the amount of suppression of other interesting sounds, such as vehicles, in the environment. The algorithm uses "templates" of different noises commonly found in urban environments to statistically extract and filter them out. PTM does not require knowledge of every source in the environment to perform noise filtering, unlike in traditional dictionary-learning methods.

Probabilistic Template Matching

The main idea behind Probabilistic Template Matching (PTM) is to generate a filter, or a set of coefficients $\alpha_i(n)$ given a window of audio, where $\overrightarrow{X}(n) = [|x(\omega_1, n)|, |x(\omega_2, n)|, ..., |x(\omega_B, n)|]^T$ is the magnitude of the time-frequency representation of time window $n$, such that the probability of the filtered window $\overrightarrow{Z}_\Lambda(n)$ being an instance of a noise of class $c_0$ is minimized. The definitions of our inputs ($\overrightarrow{X}(n)$) and outputs ($\overrightarrow{Z}_\Lambda(n)$ and $\alpha_i(n)$) are summarized next.

$$\overrightarrow{X}(n) = [|x(\omega_1, n)|, |x(\omega_2, n)|, ..., |x(\omega_B, n)|]^T$$

$$\Lambda_n = diag(\alpha_1(n), ..., \alpha_B(n))$$
\[ \overrightarrow{Z}_\Lambda(n) = \Lambda_n \overrightarrow{X}(n) \]

\( B \) refers to the number of frequency bins in our time-frequency signal representation. The \( \text{diag} \) operator creates a diagonal matrix of size \( B \times B \), where the off-diagonal entries are all zero and the diagonal entries are filled with the filter coefficients, \( a_1(n), \ldots, a_B(n) \).

We first make the assumption that the loud noise that we wish to filter out, \( c_0 \), can be described by a "template" represented by a Gaussian distribution:

\[ c_0 \sim N\left(\overrightarrow{\mu}_{c_0}, \Sigma_{c_0}\right) \]

From this assumption, an observed signal containing noise \( c_0 \) at timestep \( n \) is generated by drawing a \( B \) dimensional vector from \( N\left(\overrightarrow{\mu}_{c_0}, \Sigma_{c_0}\right) \). This vector is the time-frequency representation of the noise \( c_0 \) at timestep \( n \), where each dimension corresponds to a different frequency component of the noise. The noise corrupted signal, \( \overrightarrow{X}(n) \) would then be generated by adding in the other unknown signals (e.g. vehicle) from the environment. If \( c_0 \) has high energy over most other sounds in the environment, then the probability that our observed signal \( \overrightarrow{X}(n) \) is an instance of noise class \( c_0 \), \( P\left(\overrightarrow{Z}_\Lambda(n)|c_0\right) \), will be very high. Our goal is to generate filter coefficients \( a_i(n) \) that will reduce this probability.

However, if we minimize this probability without any constraints, all coefficients will tend to 0, cancelling out all sounds in the environment. To avoid this we introduce a novel constraint, yielding the following optimization problem shown in Equation 3.1.

\[
\begin{align*}
\arg\min_{a_1, \ldots, a_B} & \quad P\left(\overrightarrow{Z}_\Lambda(n)|c_0\right) \\
\text{s.t.} & \quad D\left(\overrightarrow{Z}_\Lambda(n)||\overrightarrow{X}(n)\right) < \beta
\end{align*}
\]

\[
D\left(\overrightarrow{Z}_\Lambda(n)||\overrightarrow{X}(n)\right) = \sum_{i=1}^{B} \left( \frac{\overrightarrow{Z}_\Lambda(n)_i}{\overrightarrow{X}(n)_i} - \log \frac{\overrightarrow{Z}_\Lambda(n)_i}{\overrightarrow{X}(n)_i} - 1 \right)
\]
The idea is still to minimize $P(\overrightarrow{Z}_{\Lambda}(n)|c_0)$ as much as possible, removing out as much of noise $c_0$ from our observation. The divergence constraint $D(\overrightarrow{Z}_{\Lambda}(n)||\overrightarrow{X}(n))$ is in place to keep the amount of change between the filtered signal and the raw signal within a threshold $\beta$ so that the filter coefficients do not completely remove all sounds from the environment. We use a static divergence constraint rather than another probabilistic constraint because we cannot assume that we have models of every possible sound in the environment. Making the assumption of knowing every sound in the environment is not feasible as there is an infinite number of potential sounds that could occur in the environment. Additionally, we chose to use the Itakura-Saito divergence metric, because of its equal weight on frequency bins with low and high energy, which is favorable for audio processing applications [82].

Optimizing over this loss function using Lagrange multipliers yields Equation 3.3.

$$L = \log \left( P(\overrightarrow{Z}_{\Lambda}(n)|c_0) \right) + \lambda D(\overrightarrow{Z}_{\Lambda}(n)||\overrightarrow{X}(n))$$

(3.3)

Finally, we arrive at the gradient update for each time window by taking the partial derivatives of our loss function $L$ with respect to our filter coefficients $\alpha_i(n)$ and substituting it into the gradient update shown in Equation 3.4.

$$\alpha_i(n + 1) = \alpha_i(n) - r \frac{\partial L}{\partial \alpha_i(n)}$$

(3.4)

One subtle point to note is that the learning rate $r$ and the $\lambda$ weight term are application tunable parameters that can be used to increase or decrease noise suppression. Higher levels of suppression will remove more noise, but will also leave a higher chance of removing out non-noise sounds from the environment. Conversely a lower suppression level will remove less noise, but will also remove less non-noise sounds from the environment. Through experimentation, we found values of $\lambda = 1e^{-5}$ and $r = 1$ to consistently yield the best separation results and highest improvement in vehicle detection rate. We use these values in our evaluation and experiments in Sections 3.4 and 3.7.
Figure 3.3: (a) Plot demonstrating the extraction process of PTM with a highly confident "template". We see that if we are more confident in the frequency component of our sound source (lower variance), PTM will extract coefficients or generated "predicted values" closer to the mean of the model template. b) However, if the model of the frequency component has higher variance (e.g. we are less confident in value of the frequency component), PTM will extract coefficients further away from the mean of the template.

A visualization of the concept behind PTM is shown in Figure 3.3. In each of these four plots, we simplify our signal and models to one dimension ($B = 1$) for visualization purposes only. A probability distribution is shown in each of the four plots, corresponding to the Gaussian "template" probability distribution. Figure 3.3(a) shows a template in which we have high confidence in. This corresponds to a template with low covariance or variance. Since we are very confident in our model, PTM estimates and extracts out a noise value that is very close to the "template" mean. Figure 3.3(b) shows a template with low confidence. This means that for this dimension, we have observed values that fluctuate greatly, thus yielding a "template" with high variance. As such, PTM is more conservative and filters out less of the signal.

Figure 3.4 shows examples of a clean passing vehicle, a jackhammer, and the two sounds mixed over the air and recorded in a real-world setting. Sounds generated from moving vehicles are primarily from their engines as well as the friction of tires on the ground. As such, throughout this work, we primarily focus on detecting and localizing these sounds from an approaching vehicle. As a comparison to neural network methods, we cleaned the mixed signal using a state-of-art neural network based source separation algorithm (MMDenseLSTM [80]). We trained the
Figure 3.4: Examples of filtering results of a vehicle sound mixed over-the-air in the real world with a jackhammer sound. We played the vehicle sound (3.4(a)) and jackhammer sound (3.4(b)) through two different speakers and recorded the mixture with a 4-channel uniform circular microphone array to generate the mixed waveform (3.4(c)). We see from visually inspecting the filtered waveforms that the MMDenseLSTM neural network approach (3.4(d)) was not able to remove as much of the jackhammer noise as PTM. (3.4(e)).

MMDenseLSTM network using artificial mixtures of vehicle and construction sounds as described in Section 3.4, since it is not possible to train source separation neural networks using mixtures recorded in the real world.

We show the filtered results in Figures 3.4(d) and 3.4(e). Through visual inspection, we can see that PTM is able to recover more of the characteristics of the clean vehicle sound than the tested neural network method. A large part of the neural network’s poor real-world performance is because it is not possible to use real mixtures to train a neural network for source separation. Using artificial mixtures may not capture all of the intricacies of the mixing process over the air in a real environment. One of PTM’s strengths over neural network-based approaches is that it only requires models and data from a single sound source (e.g. the construction sound) and does not require sound mixtures (e.g. vehicle + construction sound) and the isolated cleaned signal (e.g. vehicle)
that would require us to use artificial mixtures during training. Since CSafe is a construction worker safety platform, PTM and CSafe’s overall filtering architecture will only benefit the system if it can improve vehicle detection. We present results that support this in Section 3.4.

Although we represent distributions of our frequency domain noise "templates" as Gaussians, we show in Section 3.4 that our adaptive noise filtering architecture, that utilizes PTM, can still improve vehicle detection more than state-of-art source separation methods.

As a final note, we made the assumption that the "template" of our noise $c_0$ was available. This implies a need for a noise detector that can detect the presence of the noise and can provide a correct "template" that PTM can use to filter out noise. We describe and address both of these points next.

## Noise Detection and Template Learning

One common assumption in many content-based separation works is that the sound they are trying to separate is present in the audio stream. This assumption is not always true in real and dynamic scenarios. Hence, a noise detector is required to determine whether to perform noise separation or not. The second concern is how to learn and obtain "templates" of noise to use for filtering, as described in Section 3.3.1. We incorporate a noise detector to solve both the requirement of detecting the presence of noise in the environment and as a method for learning and providing templates required for PTM.

In general, sound event detectors operate as follows:

$$P(X \in c) > \beta$$

$X$ is the input representation of the signal (e.g. frequency spectrum), and $c$ is the class of sound we are interested in detecting. If the probability that our input observation is an example of a noise of class $c$ is greater than some threshold $\beta$, then we would detect sound $c$ in this window.

We create our noise detector in a similar fashion and choose to use a Gaussian mixture model (GMM) to model this probability distribution for each class of noise. GMMs model a probability
distribution using a linear combination of Gaussian distributions to model sub-populations within the data. Each Gaussian can be described with a mean and a covariance matrix. The mean value is the most probable value that our features will take on if our signal is indeed a sound of the specific class we are trying to detect; this is another way of saying that the *mean values of the Gaussian distributions that make our GMM noise detector can be used as templates for PTM*. The covariance is a measure of uncertainty in our template and will also be used in PTM as described in Section 3.3.1. In this way, we not only create a noise detector, necessary for intelligently applying content-based source separation, we can also leverage the way GMMs model data to provide and learn templates required for our content-based separation algorithm, PTM.

**Generalizability**

Sound source separation, like many problems that have been addressed with machine learning and deep learning, suffers immensely from lack of generalizability. When we refer to generalizability, we refer to the ability of our models and algorithms to separate out and deal with unseen examples of our target noise. We are not referring to the ability of our architecture to denoise all different types of sounds. We concern ourselves only with loud construction sounds, as these are the sounds in environment that are most likely to overpower the sound of the engines and tires of passing vehicles. Other urban noises such as passing animals or the sounds of walking are generally lower in volume and will not be as consistently present; a well-trained vehicle detector can more easily account for these scenarios than situations where, for example, a worker is continuously operating a loud jackhammer.

In CSafe, we provide pretrained models of common power tool sounds in construction sites. However, we realize that such models could not possibly account for every single type of tool. As such, we allow users the option to record sounds in the environment of their work, allowing CSafe to build template and noise detection models on the spot that are tailored to the current work environment.
3.3.2 Spatial Separation: A Filter Bank Approach

Most spatial source separation methods require the location of the sound source in advance. Since there could be multiple sources in the environment that we may need to identify (e.g., vehicles) or filter out (e.g., loud construction sounds), we first need to identify and localize these sources before performing spatial separation.

Multiple Source Localization

A common method to perform localization is to estimate the relative delay of a sound source arriving between multiple microphones in an array and use these estimates to triangulate and estimate the direction of arrival. A power-based metric, such as cross-correlation, is commonly used to accomplish this. First, the cross-correlation is computed between different microphones at different time shifts. The shift with the highest cross-correlation is estimated as the relative delay. These relative delays between microphones can then be passed into a machine learning classifier or directly used to triangulate the direction of the source. This architecture was used in a similar audio safety platform [8]. The biggest drawback in using a power-based metric and selecting the greatest peak is that it will tend to localize the loudest sound source. In construction sites, this is often the tool that the worker is operating, not the approaching vehicle. As such, it is necessary to consider methods that can estimate the location of multiple sources in the environment.

There are numerous works that introduce methods to perform multiple source acoustic localization, including MUSIC/ESPRIT and their variants [83, 84]. They all work by generating and analyzing a probability distribution of sources present at each direction of interest. This probability distribution is generated by comparing the phase difference between microphones in an array to the expected phase difference given the locations of each microphone in an array and a sound source coming from a specific direction. Once this probability distribution across all directions is generated, then a peak detection algorithm is employed to detect significant peaks and sources. The exact details and algorithms employed at each step varies from method to method.

We adopt the algorithm presented in [70] to perform multiple source localization due to a sim-
plification that reduces the computational complexity of generating the probability distribution of the presence of sources across different directions. Algorithms like MUSIC and ESPRIT generate probability distributions by taking a subspace approach, by first dividing the energy found in a single frequency, $f$, to different directions based on the probability of a source appearing at each direction. This is repeated and aggregated for every frequency of interest. Generating a probability distribution across every frequency significantly increases computation, making it difficult to use such algorithms for real-time applications. The algorithm presented in [70] makes a simplification by assigning all the energy of a specific frequency to just a single direction, which reduces computation.

Localization algorithms generally do not consider the content or the class of sounds we are localizing. This means that as long as a single sound is loud enough, our source localization module can detect and localize this source. After our filtering architecture reduces the energy of construction tool sounds, we expect to observe greater energy from vehicle sounds, which would be detected and localized. This means that our source localization module that we incorporate to localize noise sources in the environment will also double as our vehicle localization module.

**Filter Bank Spatial Separation**

Spatial separation methods such as the DUET algorithm and beamforming create filters to apply to microphone channels that diminish frequency components that are not in phase with a signal coming from the direction of the sound source [85]. Adaptive beamforming methods achieve this by continuously updating the filter based on a cost function that captures and improves certain quality measures of the received signal (e.g. SNR) [71]. To reduce computational requirements over an adaptive approach and allow for application and user tunability, we take a static approach, where we pre-generate a filter bank that we apply onto channels based on how close each frequency component aligns in phase with a signal coming from the direction of the sound source. Figure 3.5 demonstrates our proposed approach.

For every direction $d$, we generate a confined Gaussian window centered around direction $d$. 
with variance $\sigma^2$. This variance term can be used to tune the amount of suppression of other directions provided by the filter. We apply this filter by scaling the energy of each frequency in the spectrum by the filter coefficient corresponding to the direction assigned to that frequency. The direction assigned to each frequency is generated while performing source localization, as described in Section 3.3.2. Figure 3.5(a) shows an example of a filter with a higher variance and wider beamwidth, which suppresses out less energy from frequencies that do not align with the direction. While a filter with a more narrow beam and lower variance, shown in Figure 3.5(b) suppresses more energy from other directions, but may also remove more energy corresponding to
other sounds in the environment (e.g. vehicles).

For all experiments and in the CSafe wearable platform presented throughout this paper, we choose to scan across \( d = 24 \) directions divided evenly across 360°, yielding a 15° granularity. We select this parameter because it provides enough localization granularity for vehicle localization and noise separation while remaining low-cost enough to maintain real-time performance.

3.3.3 Full Filtering Architecture

In this section, we bring together all the different components proposed and introduced up until now to form CSafe’s adaptive filtering architecture for robustly filtering out common urban construction noises from the environment. Figure 3.2 shows the full noise filtering architecture of CSafe. First, we sample a window of audio from our microphone array and compute each channel’s FFT. Next, the content-based separation filters learned from our adaptive PTM algorithm during the feedback loop is then applied to clean up the audio channels. The individual channels are then provided as input to the source localization algorithm to obtain source locations. The source locations and cleaned microphone signals are then provided as input to the spatial separation module. The spatial separation module separates the individual sources in the environment. Each separated source is then passed to the noise detector to determine which sources are noise. Then, the noise sources are passed into the PTM module, where our content-based separation filters are adapted and applied to remove all detected noise in the environment, completing the loop. Sources that are not detected as noise are then fed into a vehicle detector to determine if there is a vehicle present. We use a 50 tree random forest as our vehicle detector, just like the vehicle detector used in the state-of-art audio safety platform presented here [8].

3.4 CSafe Filtering Architecture Real-World Evaluation

In this section, we compare the improvements in vehicle detection provided by CSafe’s construction noise filtering architecture, introduced in Section 3.3, with existing noise filtering algorithms and a state-of-art source separation neural network. Throughout this section, all of our
Figure 3.6: Experiment setup at a construction site.

Figures and quantitative analysis is performed on recordings of sounds in the real-world, not artificially mixed signals as is commonly done in many works presenting sound source separation methods. We also utilize two datasets for evaluation:

- **Training dataset**: Consists of 175 audio clips of common construction sounds divided into jackhammer, drilling, hammering, sanding, sawing, and vacuuming. Each clip is 10 seconds long, yielding 30 minutes of construction noise recordings. We also added an additional 5 minutes of audio clips containing vehicles passing by for a total of 35 minutes of audio. All clips were extracted from labeled YouTube clips found in the Google Audioset dataset [86].

- **Construction site dataset**: To generate this dataset we created a four microphone uniform circular array with a diameter of 15 cm, which is around the average width of a human head [87], and took audio recordings from a real construction site. Our experimental setup and construction site is shown in Figure 3.6. We recorded a total of 40 minutes of audio, during which 76 vehicles passed by. The noises prevalent in this site were jackhammering, drilling, and vacuuming sounds.
To show the improvements in vehicle detection that PTM and CSafe’s noise filtering architecture can provide, we present comparisons with 3 other algorithms: the linearly constrained minimum variance (LCMV) beamformer [71], hierarchical alternating least squares nonnegative matrix factorization (HALS NMF) [88], and the state-of-the-art source separation neural network MMDenseLSTM [80]. These algorithms were chosen as representative algorithms of a content-based separation algorithm (HALS NMF), a spatial separation algorithm (beamforming), and a neural network based algorithm (MMDenseLSTM). There is a training phase required for all of the algorithms except for the adaptive beamforming algorithm.

We trained the MMDenseLSTM network using artificially mixed construction and vehicle sounds generated from the training dataset for 12,000 epochs using a batch size of 1 clip. It is not possible to use real mixtures to train a neural network for source separation because neural networks require the exact ground truth signal to adapt weights during training. The ground truth signal in this case would be the isolated vehicle sound during the period where both sounds are occurring. However, if the sounds are both occurring at the same time, then the isolated signals of both sounds would not be available.

We trained the HALS NMF algorithm using the training dataset to learn a 50 bases dictionary of "words" to separate vehicle and construction sounds.

To gain better insight into CSafe, we evaluated three modes of operation. First, we evaluated CSafe using only the spatial separation module; we denote this as CSafe - spatial. Next, we add in the content-based separation module to see how adding this next module could improve vehicle detection. There are two modes of operation for this module, as mentioned in Section 3.3.1. In the first mode, the user does not record noises from the environment for separation and uses an existing noise detector; we denote this mode as CSafe - generic. To train this construction noise detection and separation model, we use the construction sounds from the training dataset and create a 20 mixture GMM. The second mode of operation is where the worker records a segment of the loud tool s/he will be operating to use for detection and separation; we denote this as the default CSafe mode. To train the noise detector and source separation model for this mode, we take a small 10
second segment where only the tool sound is present from clips in the construction site dataset to create a 5 mixture GMM. Since there were three periods of different tools (jackhammer, vacuum, drill), we train three models for these individual sounds and apply the corresponding model (e.g. if a jackhammer is in use, we use the jackhammer model learned from the environment).

Finally, to train and evaluate the vehicle detector, we used the construction site dataset with an 80%/20% train/test split. 56 out of the 76 recorded segments where vehicles were present were used for training and 20 segments were used for testing. All clips were divided and processed into 250ms windows with 50% overlap.

Table 3.1 shows the confusion matrix metrics for the vehicle detector under the different source separation and noise filtering schemes. The confusion matrix metrics measure the portion of 250ms windows that fall into each category. For instance, a 94% true negative rate means that the detector was able to correctly reject the presence of vehicles in 94% of windows where a vehicle was not actually present. The table also records the number of vehicles (out of the 20 passing vehicles...
used for testing) that the detector successfully detected. First, we see that the true negative and false positive rates of all the methods are relatively similar (> 90% and < 10%) respectively. This means that the detector is able to correctly identify periods where no vehicles are present very well (true negative) and does not mistakenly detect a vehicle when no vehicles are present (false positive). The differences are pronounced when we look at the true positive rates. The true positive rate is the percentage of windows where a vehicle is present in the environment and the vehicle detector is able to detect that vehicle. We see that when there is no filtering involved, the detector was only able to detect a vehicle in 80% of windows where a vehicle was actually present. We see that the LCMV beamformer, HALS NMF algorithm, and the MMDenseLSTM neural network were able to improve the detection rate to around 84%. CSafe - spatial also achieves a similar performance. This is because CSafe - spatial is using only the spatial separation module, which performs separation using similar concepts as beamforming. However, when we add in CSafe’s content-based separation module, we see noticeable improvements in the true positive detection rate. Using a pretrained model of construction tool sounds (CSafe - generic) improved the the true detection rate to 89%. Further, if the worker decides to record the tool he is using to use for construction tool filtering (CSafe), the true positive rate improves even further to 95%. This is a 15% improvement over just using a vehicle detector with no filtering. Since more windows where vehicles are present are correctly identified, CSafe also improves the number of vehicles detected as shown in the same table.

Table 3.2 further breaks down the true positive detection rate by the signal-to-noise ratio (SNR) of vehicles to the construction tool sounds. There were three prevalent construction noises in the environment: jackhammering (-8.6dB), vacuuming (-5.5dB), and drilling (1.6dB). Only one construction tool noise was present at any given moment. Though a bigger construction site may see many more tools being used at once, the microphones in the CSafe wearable platform will often observe only a single strong tool, which is the tool that the worker is currently operating or is closest to. As such, filtering out just this single tool can still account for many scenarios in construction worker safety. We obtain estimates of the SNR of these environments by taking the
Figure 3.7: The two plots show a recording of an approaching vehicle in presence of a power tool sound when: a) No noise filtering is applied, and b) CSafe’s noise filtering architecture is applied. The green segments highlight the ground truth for when a vehicle is present and the red arrows highlight the segments where the vehicle detector detects the presence of a vehicle. We see in this example, that the vehicle detector can detect the presence of vehicles earlier in each segment after applying our noise filtering architecture.

The ratio between the average power of passing vehicles when no construction sounds were present with the average power of the construction tools when no vehicles were present.

In Table 3.2, CSafe followed by CSafe - generic attained the highest true positive detection rate across all SNRs, even when the power of the construction tool (jackhammer: -8.6dB) is almost an order of magnitude higher. An interesting point is that both HALS NMF and the MMDenseLSTM network see a decrease in performance between when the SNR improves from -5.5dB to 1.6dB. This is because at 1.6dB, vehicle sounds overpower construction sounds. Blindly applying a content-based filtering technique will result in signal degradation and distortion if the construction noise is low or not present at all. The CSafe noise filtering architecture does not suffer from this problem because it includes a construction noise detector that adaptively tunes the amount of construction tool sound to filter out.

Another subtle improvement that our noise filtering architecture provides is a reduction in detection latency. We illustrate this in Figure 3.7. The signals presented in this figure is that of a vehicle sound being played on repeat and mixed in the real-world with a sound of a power tool. Figure 3.7(a) shows the raw recorded signal, while Figure 3.7(b) shows the signal after applying
our novel filtering architecture. First, we note that the peaks corresponding to the passing of the vehicle is much more noticeable in Figure 3.7(b) now that the majority of the construction sounds have been filtered out. Next, we placed each clip through our random forest vehicle detector. The segments highlighted in green are segments where the vehicle is audibly present. Segments highlighted in red show the time frames where our vehicle detector detected the presence of the vehicle. We can see that after applying our noise filtering architecture, we can detect a greater portion of the time frame that the vehicle is present. Additionally, we see that without applying our noise filtering architecture, the detector is only detecting the vehicle at its loudest point when the vehicle is passing by. Detecting a vehicle and alerting the user at this point is already too late because we need to give the user enough time to react. After applying our noise filtering algorithm, we see that the vehicle is detected much earlier before the peak when the vehicle is passing the user, giving the user much more time to react. In this example, the detector was able to detect the vehicle 90ms after the vehicle comes into audible range without applying any filtering, while the detector was able to detect the vehicle in 15ms when our noise filtering architecture is applied. This is critical in our worker safety platform, where every millisecond of reduced latency allows that much more time for the user to react to oncoming dangers.

To further quantify this improvement in latency, we repeated the steps described above for Figure 3.7 for every sample in our noisy dataset and plot the mean and variance in detection latency for each tested method in Figure 3.8. Without any filtering, our vehicle detector was able to detect vehicles with an average of 42.5ms after the vehicle comes into audible range. Each separation method was able to decrease this lag, but CSafe - generic was able to bring the average delay down to 22.5ms. CSafe’s non-generic mode was able to bring this delay even further down to 16.4ms, which is lower than any other method, allowing users that much more time to react after receiving an alert. We note that these latencies were generated using computer implementations of each algorithm while we experimented with different source separation algorithms to use in the final CSafe wearable system. A latency analysis of the full CSafe wearable platform is presented in Section 3.6.2.
Figure 3.8: Average latency of vehicle detection of our vehicle detector without applying noise filtering and after introducing noise filtering. We see that CSafe’s noise filtering architecture is able to reduce detection latency from 42.5ms, without any filtering, down to 16ms, which is a greater reduction than any of the other methods tested.

Figure 3.9: CSafe’s full system architecture spanning the embedded and smartphone sub-platforms.

3.5 CSafe Platform

In this section we introduce the CSafe wearable and smartphone platform. We also discuss our dataflow for the entire system spanning noise filtering, vehicle detection, and vehicle localization.
3.5.1 System Architecture

Figure 3.9 shows the full system architecture of CSafe spanning the embedded hardware platform and smartphone system. The hardware platform houses and samples from the microphone array. The FFT of each window sampled is then computed and these features are sent to the smartphone platform via Bluetooth Low-Energy (BLE), a low-power wireless transmission protocol. The features from the microphone array are then passed onto our novel adaptive noise filtering architecture (Section 3.3.3) that filters and removes urban and construction noise sounds from our signals. The outputs from this module are the filtered non-construction sound sources currently present as well as their current location with respect to the user. These sources are then passed to our vehicle detector to determine if any of the sources are vehicles. We use a 50 tree random forest detector, just like the vehicle detector employed by this [8] audio safety platform for pedestrian safety. Finally, an alert containing the direction of vehicles nearby is sent to the user if any of the sources are detected to be vehicles.

3.5.2 Embedded Hardware Platform

The CSafe embedded hardware platform was shown in Figure 3.1. We integrated the same embedded circuit consisting of a Cortex-M4 microcontroller and BLE module, presented in this work on pedestrian safety [8], into CSafe along with an array of four low-power MEMS microphones. These components are integrated onto a construction helmet, with the total cost of the major electrical components coming out to less than 20USD. We note that though the embedded circuit is the same, almost every other aspect, from the architecture and algorithms is novel because of the unique challenges present in construction worker safety.

The embedded hardware platform samples from the four microphone channels and computes the FFT for windows of 250ms with 50% overlap for each channel. These features are then transmitted to the smartphone over BLE. This provides enough granularity for CSafe to reliably detect vehicles with low latency while satisfying BLE’s bandwidth limitations.

For this work, we embed our hardware platform into a helmet commonly worn by construction
workers, but note that our platform can be easily incorporated into many other kinds of wearables, such as hats and headphones, for various kinds of users.

3.5.3 Smartphone Platform

Once the smartphone platform receives the frequency spectra of the microphone array, the smartphone executes our novel adaptive noise filtering architecture, introduced in Section 3.3.3. The output to the noise filtering architecture is the separated sound sources and corresponding locations. The outputs from the noise filtering architecture are then passed onto the vehicle detector. If any of the separated sources are detected as vehicles, a visual alert is sent to the user through the smartphone system indicating the direction and distance of the vehicle from the user. Since construction workers are often busy working and may not be able to look at their phones, we also send an audio alert to the user’s headset/audio-enabled ear protection and provide haptic feedback through the smartphone system.

We decided to leverage the smartphone platform to execute most of the algorithms found on CSafe, as shown by the uneven partition of computation between the embedded hardware platform and the smartphone system in Figure 3.9. This is because there is a low amount of computational resources available on our Cortex-M4 based hardware platform when compared to much stronger processors available on modern-day smartphones. Additionally, we allow users to record audio clips of urban and construction noise in the immediate environment to generate models that are tailored to the current environment, allowing our architecture to more robustly filter construction tool sounds.

3.6 System Evaluation

3.6.1 Power Consumption

’s embedded hardware platform sees a 69 mA current draw off of a 3.3V power source. This allows to run for 14.5 hours off of two standard AAA batteries, connected in series, each with a 1000 mAh capacity, before recharging. This duration is more than enough for frequent daily use.
3.6.2 Latency

We measured the execution time of every component in our data pipeline, as shown in Figure 3.10. CSafe first samples 250ms windows of audio from its microphone array with 50% overlap. This means that CSafe calculates and transmits features every 125ms. The feature computation and wireless transmission to the phone takes 100ms and 3ms to execute, respectively. The CSafe smartphone system executes its entire pipeline, including the noise filtering pipeline, vehicle localization + detection, and sending user alerts in less than 8ms. This yields a full end-to-end latency, from when CSafe begins sampling a window to when CSafe is able to send vehicle presence and location alerts to the user, of 236ms. This is on par with the average human reaction time, allowing users enough time to react to oncoming vehicles.
3.7 CSafe Platform Real-World Evaluation

In this section, we evaluate CSafe through a series of real-world experiments. We evaluated two aspects of CSafe: vehicle detection accuracy and localization accuracy. We evaluated CSafe in the same environment shown in Figure 3.6. To obtain the ground truth vehicle presence and location, we record all scenarios with an additional video recorder and sync the recorded video and audio with output logs from CSafe. In all experiments we compare CSafe with the state-of-art PAWS [8] audio safety platform, developed for pedestrian safety. In the following experiments, we analyzed 30 vehicles that passed by for the PAWS system. We also analyzed 30 vehicles each for CSafe and CSafe - generic.

3.7.1 Vehicle Detection

Both CSafe and CSafe - generic detected a high percentage of vehicles (29 out of 30 and 30 out of 30, respectively). The PAWS system was only able to detect 21 out of 30 vehicles. Table 3.3 compares the confusion matrix metrics for vehicle detection of CSafe and PAWS. The confusion matrix metrics list the percentage of audio frames, rather than vehicle counts, that were categorized as a true positive, true negative, false positive, and false negative.

We see that for CSafe, both true positive and true negative rates are very high (CSafe - generic: 78% and 99%; CSafe: 82% and 96% respectively), while PAWS has a much lower true positive rate (66%) because of the presence of the loud construction tools obfuscating the sound of the oncoming vehicles. CSafe improves the true positive vehicle detection rate over PAWS by 16%.

The usability of the system is greatly affected by the false positive rate, which is the percentage of windows where a vehicle is not present, but the detector incorrectly detects a vehicle. If the false positive rate is too high, a user may become annoyed and less likely to heed the alerts of the system later on. We see that the false positive rates of both PAWS and CSafe are relatively low.

Finally, the false negative rate should be very low to avoid missing too many vehicles, which could be life threatening. We see that due to the added overpowering noise of construction tools
Table 3.3: Confusion matrix metrics comparing both modes of CSafe with a state-of-art pedestrian safety system [8]

<table>
<thead>
<tr>
<th></th>
<th>True Pos.</th>
<th>True Neg.</th>
<th>False Pos.</th>
<th>False Neg.</th>
<th>Vehicles Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSafe</td>
<td>82%</td>
<td>96%</td>
<td>4%</td>
<td>18%</td>
<td>29/30</td>
</tr>
<tr>
<td>CSafe - Generic</td>
<td>78%</td>
<td>99%</td>
<td>1%</td>
<td>22%</td>
<td>30/30</td>
</tr>
<tr>
<td>PAWS [8]</td>
<td>66%</td>
<td>99%</td>
<td>1%</td>
<td>34%</td>
<td>21/30</td>
</tr>
</tbody>
</table>

Table 3.4: Localization error comparison between CSafe and a state-of-art pedestrian safety system, PAWS [8]

<table>
<thead>
<tr>
<th></th>
<th>Avg. Error (degree)</th>
<th>Std. Dev. Error (degree)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSafe</td>
<td>6.90°</td>
<td>5.70°</td>
</tr>
<tr>
<td>CSafe - Generic</td>
<td>11.30°</td>
<td>10.07°</td>
</tr>
<tr>
<td>PAWS [8]</td>
<td>38.7°</td>
<td>18.60°</td>
</tr>
</tbody>
</table>

and its lack of a mechanism to deal with this unexpected noise, PAWS’s false negative rate is very high, at 34%. Because of their noise filtering mechanisms to filter out the loud construction sounds in the environment, both modes of operation for CSafe achieve a much lower false negative rate of around 20% or lower.

Overall, the high true positive rate and low false negative detection rates contribute significantly to CSafe’s improved performance over PAWS as an audio safety platform suitable for construction worker safety and other scenarios beyond pedestrian safety.

3.7.2 Vehicle Localization

Table 4.7 compares the average direction of arrival localization error between CSafe, CSafe - generic, and PAWS in degrees. We see that the average error rate of PAWS is much higher than that of both modes of CSafe despite PAWS using a localization algorithm with more granularity. This is because PAWS uses a cross-correlation based method to estimate relative delays between microphones in its array. As mentioned in Section 3.3.2 these methods are only capable of capturing a single source in the environment. Most of the time, the source that is captured is the loudest sound in the environment as the sound with the highest energy influences the values of the cross-correlation function the greatest. Since the construction tool is often the loudest sound in
the environment, rather than the approaching vehicle, PAWS will tend to localize the sound of the construction tool rather than the vehicle, leading to high localization errors. On the other hand, CSafe’s novel adaptive noise filtering architecture is able to filter out most construction sounds and localize multiple targets in the environment. This reduces the effect of the overpowering construction noises in the environment on the detection and localization of oncoming vehicles, which leads to higher localization accuracy.

3.8 Audio Privacy

We have seen that by incorporating our CSafe adaptive noise filtering architecture to filter out construction sounds, our audio wearable platform could better detect vehicles. In this section, we explore how our adaptive noise filtering architecture could be used to improve privacy, by filtering out speech, in mobile audio applications.

Smartphones have greatly impacted our daily lives, providing easy ways to monitor different aspects of our health, entertain us, and much more. In 2019, more than 81% of Americans owned a smartphone, which was up from 35% in 2011 [89]. Engagement in mobile applications has also increased. In 2008, Americans spent only 20 minutes per day on mobile applications, compared to more than 3 hours in 2016 [90]. Additionally, the number of smartphone application downloads is expected to increase from 178 billion in 2017 to 258 billion by 2022, and the total revenue generated from these applications is expected to grow from 88 billion USD in 2016 to more than 180 billion USD by the end of 2020 [91]. The increase in smartphones and smartphone usage has also spurred on numerous mobile wearable platforms for various applications such as safety [de2018paws, 11, 8, 6, 9] and health monitoring [92, 93].

As the number of smartphone users and applications increase, one of the biggest concerns is privacy and security, especially when it comes to applications that "listen" to our surroundings. In 2019, a dutch news outlet (VRT NWS) obtained more than 1,000 recordings collected through Google Home and Assistant applications and found that more than 150 of the clips were recorded despite the lack of the "OK Google" command [94]. In other words, more than 10% of recordings
should never have been made, demonstrating a huge audio privacy and security risk. A recent report from The New York Times revealed that around 1,000 smartphone application use software that is known to listen to TV signals to track viewership behavior, often without knowledge from the smartphone user [95].

We introduce PAMS, a software development package for enhancing privacy and security in audio-based mobile applications. PAMS allows developers and users to apply a set of privacy filters on audio recordings in their own applications to filter out sounds and noises in the environment that are not required, thereby reducing the amount of additional sensitive signals saved. To accomplish this, we integrate and adapt our Probabilistic Template Matching algorithm, used in our CSafe adaptive noise filtering architecture, to filter and remove privacy-sensitive noises from the environment, namely speech.

We demonstrate the effectiveness of PAMS through a mobile sleep monitoring system. Sleep quality monitoring applications rely in large part on raw recordings of the microphone to observe and analyze breathing, snoring, and other sounds during sleep. However, there could be sensitive sounds in the environment, such as speech or neighborhood sounds, that the microphone can record that should not be recorded in the first place. We show that by applying PAMS to a custom sleep monitoring system, we can reduce speech recognition accuracy by up to 74.3% compared to other sleep monitoring applications that are freely available while maintaining a similar snoring event detection performance.

3.8.1 Related Works

There are a few sleep monitoring works in the literature that primarily use audio as a means to measure sleep quality. In general, these works extract a set of features from windows of audio, such as mel-frequency cepstral coefficients (MFCCs), empirical mode decomposition features (EMD), and autocorrelation. These features are then passed to an acoustic event classifier, such as a k-nearest neighbors classifier (KNN), that is trained to determine if snoring or breathing sounds are present [96, 97, 98, 99, 100]. [101] takes a similar approach in using audio to determine if
the patient suffers from sleep apnea. However, rather than observing breathing sounds while the person is asleep, they use speech recordings taken from the person while he is awake. In all these works, raw audio is recorded and analyzed in strict lab settings. In non-lab settings, recording and saving raw audio poses a privacy risk, as there may be other sensitive sounds, like speech, in the environment that the microphone records. These works do not account for this privacy issue.

There are also numerous sleep monitoring smartphone applications out on the market, meant for in-home and non-lab use. Sleep as Android [102], SnoreLab [103], and Sleep Cycle [104] are just a few examples. Though these applications all have the capability of using more sensors to estimate sleep quality, one of the main sensors most of these applications use is the microphone. Users start the application as they go to sleep, and the application records their acoustic environment throughout the night. These applications will record and save all sound segments where the signal power is above a certain threshold regardless of the content of the signal. These signals are then used in conjunction with other sensors of the application that the user enables to provide sleep quality analysis. Since these applications record all sounds in the environment above a certain power threshold, they can record speech or other privacy-sensitive sounds that may be present. To the best of our knowledge, none of the available commercial and freely available smartphone sleep monitoring applications have mechanisms to account or filter out these privacy-sensitive signals.

To help ensure user privacy in sleep monitoring and other audio-based monitoring applications, we take a sound source separation approach to filter out privacy-sensitive signals, namely speech, from the environment, by incorporating Probabilistic Template Matching.

3.8.2 PAMS Pipeline

Figure 3.11 shows the architecture and pipeline for PAMS. Audio from the microphone passes to the PAMS module, where the signal is transformed into the frequency domain. Then, the PTM privacy filters are applied to the signal to filter out the privacy-sensitive sounds. The output of the filtering process can then be further processed or analyzed depending on the application. This cleaned signal is also fed into the noise detector which is then used to update PTM privacy filters.
The filter coefficients, $\alpha_i$, are updated based on the speech templates provided by a Gaussian Mixture noise model and applied in the next window.

The updated filters are then applied to the next window and process repeats.

### 3.8.3 Sleeping Event Detection

Before we introduce the system architecture for our PAMS enhanced sleep monitoring smartphone system, we will first introduce our sleeping event detector. Most sleep monitoring smartphone applications will record and analyze sounds using a simple volume-based detector. If the power-level or volume of the audio that is observed at the microphone is above a certain threshold, then the application records and saves the sound. Otherwise, the sound is discarded. We adopt a similar approach for detecting and saving sleep recordings.
3.8.4 Audio-Based Sleep Monitoring Mobile System Architecture and Design

Figure 3.12 shows the system architecture and data flow for our PAMS enhanced sleep monitoring system. PAMS samples 250ms windows with 50% overlap and computes the magnitude spectrum of the window. This means, that the pipeline is executed every 125ms. In this application, we wish to reduce the amount of speech that can be recorded. As such, we then apply our privacy filters learned from the adaptive PTM algorithm to filter out speech. As mentioned in Section 3.8.2, the cleaned signal is then passed to the noise detector. The noise detector in this case is a speech detector, which determines how much speech is left within the signal and is used to update the privacy filters to apply to the next window of audio. The cleaned signal is also processed by the sleep sound event detector, mentioned in Section 3.8.3 to determine whether any heavy breathing, snoring, or other sleep sounds are present. If the detector detects sleeping sounds, then the window is saved as a recording. Otherwise, the window is discarded, much like what is done in existing sleep monitoring smartphone systems.

The entire sleep monitoring pipeline runs in less than 5ms on a Samsung Galaxy S8, which is less than the pipeline execution period of 125ms, meaning that the PAMS pipeline alone is also capable of running in real-time.

The sleep monitoring pipeline shown in Figure 3.12 is implemented on an Android device (Samsung Galaxy S8). Users are able to monitor their sleep, listen to recordings, and provide speech samples to PAMS to generate privacy-preserving speech filters through the user application.

3.8.5 Evaluation

We evaluate our PAMS enhanced sleep monitoring system based on two measures: privacy and preservation. In this context, privacy refers to the idea that if speech is present during sleep, then the final recordings should contain little to no speech. However, if the filters remove too much of the signal, it may also remove parts of the signal that is useful our application. For sleep monitoring, this would be snoring, breathing, and other sleeping sounds. An application that has high preservation should still be able to capture these necessary sounds even if we are processing
Table 3.5: Speech recognition accuracy of recorded clips from PAMS and Sleep as Android [102]. This table lists the proportion of words correctly identified in the recorded clips, the proportion of words incorrectly identified, and the proportion of words that were not even detected by Google Speech to Text [105].

<table>
<thead>
<tr>
<th></th>
<th>Correct</th>
<th>Incorrect</th>
<th>Not Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAMS</td>
<td>21.3%</td>
<td>5.1%</td>
<td>73.6%</td>
</tr>
<tr>
<td>Sleep as Android</td>
<td>95.6%</td>
<td>4.0%</td>
<td>0.4%</td>
</tr>
</tbody>
</table>

Figure 3.13: Confusion matrix metrics for the sleep event detectors of Sleep as Android and PAMS.

altering the signal. We compare against Sleep as Android [102], a freely available sleep monitoring smartphone application on the Google Playstore.

We randomly selected 5 clips of snoring sounds from the Google Audioset dataset [86]. In order for PAMS to learn the voice model of the speaker, we had each speaker read a randomly chosen article from Wikipedia [106] and recorded a 20 second segment to use as training. To generate test recordings, we ran our PAMS enhanced sleep monitoring system and Sleep as Android while playing one of the randomly chosen snoring clips and had a speaker read from a different passage.
than what was used to generate the voice model. In this way, both our PAMS enhanced system and Sleep as Android recorded snoring sounds mixed with speech. In total we repeated this procedure for 4 speakers, 3 passages, and 5 snoring clips, for a total of 60 mixed clips, each around 20 seconds long.

To evaluate privacy and speech intelligibility, we ran the recorded clips from Sleep as Android as well as the filtered clip produced by PAMS through Google Speech-to-Text [105]. Table 3.5 shows the proportion of correctly identified words from both Sleep as Android and PAMS. The table lists the portion of words correctly transcribed (correct), the portion of words incorrectly transcribed (incorrect), and the portion of words that were never even detected by Speech-to-Text (not detected) out of a total of 3,468 words spoken across all recordings. Google Speech-to-Text is able to correctly transcribe 95.6% of the words spoken from each user. Speech-to-Text is also able to decipher most of the spoken words, as shown by the low "not detected" percentage. On the other hand, PAMS shows a much lower correctly transcribed rate, at 21.3 percent. This is a huge difference of 74.3 percent. Additionally, PAMS boasts a much higher "not detected" rate. This is because of the PTM speech filtering algorithm that PAMS employs to eliminate speech before the signal is saved. On the contrary, Sleep as Android has no processing module to remove privacy-sensitive non-sleep sounds from the acoustic environment.

To evaluate preservation, we ran the recorded audio windows through the sleep event detectors in both systems. Figure 3.13 shows the confusion matrix metrics for the Sleep as Android and the PAMS enhanced sleeping event detectors. The percentages refer to the portion of 250 ms windows where sleep sounds were detected by Sleep as Android and PAMS. The true positive rate is the percentage of windows where a breathing or snoring sound was present, and the system correctly captured and saved that window. A system that is able to reliably detect and analyze snoring and sleep sounds should have a high true positive rate. We see that Sleep as Android (99 percent) and PAMS (93 percent) yielded similar strong performances in this metric. This means that even after processing the audio stream with no knowledge of breathing and snoring sounds used in the filtering process, PAMS is still able to capture and detect snoring sounds as well as a smartphone

91
system that is freely available. Related to the true positive rate is the false negative rate, which is the portion of windows where snoring or breathing is present, but the detector incorrectly identifies the window as not containing snoring or breathing sounds. Both Sleep as Android (1 percent) and PAMS (7 percent) show similar performance here.

The false positive rate is the portion of windows that do not contain breathing or snoring sounds, but the system detects and records it anyways. This rate should be very low to avoid using sounds that are not related to sleeping as an estimate of sleep quality. Sleep as Android (93 percent) boasts a much higher false positive rate than PAMS (57 percent). Sleep as Android, like many other freely available applications uses a simple power-based metric to determine when to record and analyze audio to measure sleep quality. Since voice is generally high volume, it will likely record and analyze voice. On the other hand, PAMS avoids recording many of the windows that contain speech because of the PTM speech filtering pipeline that is first employed to filter out speech. Related to the false positive rate is the true negative, which is the portion of windows that do not contain breathing or snoring sounds that is correctly identified as non-sleeping sounds. This metric should be high in order to avoid using non-sleeping sounds to measure sleep quality. We see that Sleep as Android (7 percent) has a much lower true negative rate than PAMS (43 percent) because speech is present in most of the windows where snoring or breathing is not present. Due to the simple power-based metric used to generate recordings, the high-powered speech signal will cause Sleep as Android to record speech even when no breathing is present. PAMS is able to reliably filter out speech, allowing its detector to more reliably reject intervals of speech as non-sleep events.

We note that the true negative rate of 43% for the PAMS enhanced sleep detector is by no means intended to represent the state-of-art performance. We are instead highlighting the notion that applying PAMS not only improves privacy, but can also significantly improve certain performance measures in other areas of the application (e.g. the large improvement in true negative rate from 7% to 43%).
Chapter 4: A Common Selective Audio Filtering Framework for Mobile, Embedded, and Cyber-Physical Systems

As we have seen in Chapter 3, we were able to extend our audio-based wearables for pedestrian safety to construction worker safety by incorporating a real-time adaptive construction tool filtering architecture to filter out construction sounds and reveal vehicle sounds. We realize that there is a wide range of applications that could benefit from audio filtering, beyond urban safety. Because the goal is to create an architecture that can account for a wide range of applications, we need to account for a wide range of different signal representations (e.g., raw signal vs. MFCCs) and sound model types (e.g., support-vector machine vs. mixture of Gaussians). Different types of applications may benefit from different signal representations or different sound models, so our architecture should be able to account for a wide selection. Additionally, we also need to incorporate a mechanism that allows developers to select sounds to enhance, not only filter out. These requirements, in addition to general systems requirements (e.g., power and latency), make this problem very challenging. In this chapter, we introduce a common selective filtering framework that addresses all of these challenges and can be easily embedded into a wide range of mobile, embedded, and cyber-physical applications.

we introduce AvA, an Adaptive Audio filtering architecture for enhancing different types of sounds on a wide range of systems [107]. In many acoustic systems, developers create models of sounds that need to be detected or filtered out. For instance, a smartphone may have a command phrase detector to determine when a command phrase is spoken and a model for speech to determine what was spoken. AvA allows users to choose which sound types to either filter out or enhance by directly leveraging the sound models that developers create for their specific application. As such, AvA is adaptable to a wide range of different sound detectors and signal features.
AvA accomplishes this by incorporating content-informed adaptive beamforming (CIBF), a novel adaptive beamforming algorithm that directly incorporates sound detectors to learn filter coefficients to better detect or filter out specific sounds. CIBF leverages the advantages of both spatial filtering and content-based filtering to outperform methods that only use either spatial filtering (i.e. BSS) or content-based filtering in non-artificially mixed scenarios. CIBF enables AvA to account for a wide range of different sound models and signal feature representations using a novel three step approach (model adaptation, feature adaptation, and signal adaptation). AvA’s adaptability to a wide range of signal features, machine learning models, and low-resource systems allows us to more easily embed acoustic intelligence anywhere and impact many areas such as wearables [108, 109, 92, 110, 111], built environments [112, 113, 114, 115, 116, 117, 118], and health [119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129]. We summarize the major points of this chapter below:

• We propose AvA, a novel acoustic filtering architecture that adaptively filters out or enhances different sounds depending on application needs. AvA accomplishes this by directly incorporating sound models, a developer may have already created for an application, to filter out or improve detection.

• We propose content-informed adaptive beamforming (CIBF), a novel adaptive beamforming algorithm that uses a novel three step approach (model adaptation, feature adaptation, signal adaptation) to learn filter coefficients to filter out or improve detection based on user supplied sound models. AvA leverages CIBF to be adaptable to a wide range of different sound models and signal representations.

• We demonstrate through four scenarios, three different model types, and two different features the capability of AvA in enhancing or filtering out different types of sounds in a wide range of scenarios and configurations, thus highlighting the generalizability and customizability of AvA. Across these scenarios, we show that that AvA outperforms state-of-art filtering algorithms, improves target detection performance by up to 11.1%, and reduces noise
detection by up to 78.9%.

- We perform two case studies, where we integrate AvA into two mobile/embedded platforms to show the adaptability of AvA. We compare the performance of the AvA-enhanced systems against existing state-of-art systems and show how AvA can boost detection performance in real applications.

4.1 Related Works

There are numerous mobile and embedded applications that leverage audio. Audio-based systems have been deployed for numerous applications including, but not limited to, gunshot detection [130], vehicle detection and localization for urban safety [de2018paws, 8, 11, 10, 9], activity detection [131, 132], robotic intelligence [133], and much more [134, 135]. Many of these works focus on the design of classifiers to achieve the best performance [136]. [137] presents a cloud-based system for acoustic event detection that uses user-contributed sound clips to train acoustic detectors for specific mobile applications. Instead, we take an acoustic filtering approach to remove or enhance sounds in the environment depending on application needs.

There are two broad categories of filtering algorithms: spatial filtering and content-based filtering. Spatial filtering methods use multiple observations in space by placing microphones at different locations to perform filtering. Methods that fall into this category include, but are not limited to beamforming [70, 71, 72], blind source separation (BSS) [75, 85], and two microphone filtering techniques [138, 73]. These methods do not incorporate the content or the types of sounds present in the environment and generally require the location of sources beforehand to perform filtering.

Content-based filtering methods generally require only one microphone. These methods, such as deep neural networks (DNN), use trained models of specific sounds to filter them out [139, 140, 141]. Because they are trained to deal with specific sounds, applying a model trained in one context to a different application may significantly degrade our signals. In this regard, unlike spatial filtering methods, content-based filtering methods are not agnostic to the sound types present in
the environment. In this work, we propose content-informed adaptive beamforming (CIBF), a novel adaptive beamforming algorithm that bridges the gap between spatial and content-based filtering, leveraging the strengths of both types of filtering. CIBF allows AvA to be a powerful tool for enhancing or filtering out sounds that a developer has trained a model for (content-based filtering), while providing a content-agnostic way of filtering sounds we do not have models for (spatial filtering).

[12] proposes an acoustic wearable system for detecting and localizing vehicles to improve construction worker safety. This work proposes an adaptive filtering architecture that improves vehicle detection by filtering out construction site sounds. However, the architecture is specific to the proposed acoustic wearable, limited to only filtering construction sounds, and supports only a single signal feature representation (power spectrum) and sound model (mixture of Gaussians). In this work, we propose AvA and CIBF, which can both enhance and filter signals while supporting a wide range of different sound models and signal representations.

### 4.2 Content-Informed Adaptive Beamforming

We propose content-informed adaptive beamforming (CIBF), a novel adaptive beamforming algorithm that directly incorporates acoustic detection and sound models to improve detection performance. Users and applications can select different sounds to either improve or degrade detection performance depending on application needs. CIBF supports a wide range of different sound models, classifier types, and frequency-domain signal representations. A typical problem set up for beamforming is shown next.

\[
\arg \min_{w(t,f)} L(w(t,f), x(t,f)) \\
\text{subject to } w^*(t,f)d(f) = 1
\]  

(·)^* and (·)^T are the conjugate and regular transpose operators, respectively. \(x(t,f)\) is the vector of observations from each of the \(n\) microphones at time window \(t\) and frequency \(f\), where \(x(t,f) = [x_1(t,f), x_2(t,f), ..., x_n(t,f)]^T\). \(x_i(t,f)\) is the short-time frequency representation of the signal.
from microphone \(i\) at time step \(t\) and frequency \(f\). \(\mathbf{w}(t, f) = [w_1(t, f), w_2(t, f), \ldots, w_n(t, f)]^T\) is the vector of filter coefficients applied to each of our \(n\) microphone observations at each frequency and time step. \(\mathbf{d}(f) = [d_1(\theta, f), d_2(\theta, f), \ldots, d_n(\theta, f)]^T\) is the steering vector that depends on the steering direction, \(\theta\). Beamforming attempts to adapt a set of filter coefficients \(\mathbf{w}(t, f)\) to retain signals arriving from steering direction \(\theta\), while attenuating signals arriving from other directions. This is accomplished by the direction constraint, \(\mathbf{w}^*(t, f)\mathbf{d}(f) = 1\), and the choice of loss function. In this work, we use the commonly used linearly constrained minimum variance (LCMV) loss function shown below [71].

\[
L(\mathbf{w}(t, f), \mathbf{x}(t, f)) = \mathbf{w}^*(t, f)E[\mathbf{x}(t, f)\mathbf{x}^*(t, f)]\mathbf{w}^*(t, f)
\]

\(E[\cdot]\) is the expectation operator. We see that the filtering process depends solely on the steering direction (i.e. sound source direction). Although enhancing our signal in this way may improve the signal-to-noise ratio, it is not guaranteed to improve or reduce detection.

In CIBF, we incorporate sound models and acoustic classifiers. In general, an acoustic detector analyzes a signal holistically and determines that a sound of class \(c\) is present in environment if 
\(P_c(F(s(t))) > a\), where \(s(t) = [s(t, f_1), \ldots, s(t, f_{n_B})]^T\) is the frequency domain representation of an acoustic signal and \(n_B\) is the number of frequency bins in our signal. Typically, traditional machine learning classifiers do not operate directly on the the raw signal, but rather on a set of extracted features. We refer to the operation \(F(x(t))\) as the set of extracted features from the raw signal, \(x(t)\). Any detector for sound \(c\) evaluates a decision function \(P_c(\cdot)\) to determine whether the input is an instance of sound \(c\). If this function is greater than some defined threshold \(a\), then the model will detect the presence of sound \(c\).

One way to to filter out sound \(c\), or prevent \(c\) from becoming detectable, is to ensure that the filtered signal remains below the detectable threshold, \(a\). That is to say, we should learn a set of coefficients, \(\mathbf{w}(t, f)\), such that 
\(P_c(F(D(W^*(t)X(t)))) < a\). \(D(\cdot)\) is the diagonal operator that returns the diagonal entries of a matrix as a vector. The matrices, \(W(t)\) and \(X(t)\), are shown next:
\[ W(t) = \begin{bmatrix} w(t, f_1) & \ldots & w(t, f_{n_B}) \end{bmatrix} \]

\[ X(t) = \begin{bmatrix} x(t, f_1) & \ldots & x(t, f_{n_B}) \end{bmatrix} \]

\[ D(W^*(t)X(t)) = \begin{bmatrix} w^*(t, f_1)x(t, f_1) \\ \vdots \\ w^*(t, f_{n_B})x(t, f_{n_B}) \end{bmatrix} \]

\( W(t) \) and \( X(t) \) are formed by concatenating the filter coefficient vectors, \( w(t, f) \), and signal vectors, \( x(t, f) \), across all frequencies. In other words, the filtered signal, \( D(W^*(t)X(t)) \), is obtained by applying each filter coefficient vector, \( w(t, f) \), to the corresponding signal vector, \( x(t, f) \), at each frequency.

On the contrary, if we wish to "enhance" or improve our detection rate of sound \( c \), we should learn a set of coefficients such that our filtered signal remains above the detectable threshold. That is to say: \( P_c(F(D(W^*(t)X(t)))) > a \). For clarity, we denote the the filtered signal throughout the rest of the paper as \( Z_t = D(W^*(t)X(t)) \). The full CIBF problem setup is shown in Equation 4.2.

\[
\arg\min_{w(t,f)} L(w(t,f), x(t,f)) \\
 w^*(t, f) d(f) = 1 \\
P_{e_i}(F(Z_t)) > a_{e_i}, 1 \leq i \leq n_e \\
P_{f_j}(F(Z_t)) < b_{f_j}, 1 \leq j \leq n_f
\]

We refer to \( P_{e_i} \) as the decision function of sound \( e_i \) that the user wants to enhance, while \( P_{f_j} \) refers to the the decision function of sound \( f_j \) that the user wants to filter out. \( n_e \) and \( n_f \) refer to the total number of sound types a user wishes to "enhance" or "filter out", respectively. We summarize the constraints of CIBF next.

- **Direction Constraint:** \( w^*(t, f) d(f) = 1 \)
• Enhancement Constraints: $P_{e_i}(F(Z_t)) > a_{e_i}$

• Filtering Constraints: $P_{f_j}(F(Z_t)) < b_{f_j}$

We attempt to solve this problem with Lagrange multipliers ($\lambda$’s), shown in Equation 4.3:

$$L_{\lambda}(w(t, f), x(t, f)) = L(w(t, f), x(t, f))$$
$$- \lambda_d(w^*(t, f)d - 1)$$
$$- \sum_{i=1}^{n_e} \lambda_{e_i}(P_{e_i}(F(Z_t)) - a_{e_i})$$
$$+ \sum_{j=1}^{n_f} \lambda_{f_j}(P_{f_j}(F(Z_t)) - b_{f_j})$$

(4.3)

It is difficult to directly solve for the multipliers for each constraint, given the wide range of models and features that can be used. As such, we take a gradient moving in the direction of the negative gradient at each iteration, as shown in Equation 4.4.

$$w(t + 1, f) = w(t, f) - \epsilon \nabla_{w(t)} L_{\lambda}(w(t, f), x(t, f))$$

(4.4)

Here, $\epsilon > 0$ is the step size. Due to the various configurations of classifiers and features a sound or detection model can use, it is difficult to choose multipliers that satisfy all of the enhancement and filtering constraints in Equation 4.2. As such we only implicitly eliminate the direction constraint from the optimization problem by solving for $\lambda_d$ in terms of the other constraints. Applying the direction constraint to Equation 4.4, solving for $\lambda_d$ in terms of the enhancement and filtering multipliers ($\lambda_{e_i}$ and $\lambda_{f_j}$), and substituting this value back into Equation 4.4 yields the final CIBF update shown in Equation 4.5. For clarity, we denote $w(t, f) = w(t)$ and $x(t, f) = x(t)$, and $I$ is the identity matrix. One assumption present in the Equation 4.5 is that our system does not have an estimate of the spatial correlation matrix, $E[x(t)x^*(t)]$. This is because the environment and types of sounds may be time-varying and changing frequently. As such, we make the simple,
but common, estimation of $E[x(t)x^*(t)] = x(t)x^*(t)$, and denote the output of CIBF (i.e. the "beamformed" signal) as $y(t) = w^*(t)x(t)$.

$$w(t + 1) = w(t) + d(d^*d)^{-1}[1 - d^*w(t)]$$
$$- \varepsilon[I - d(d^*d)^{-1}d^*]x(t)y(t)$$
$$- \varepsilon[I - d(d^*d)^{-1}d^*]\sum_{j=1}^{n_f} \lambda_{fj} \nabla_{w(t)} P_{f_j}(F(Z_t))$$
$$+ \varepsilon[I - d(d^*d)^{-1}d^*]\sum_{i=1}^{n_e} \lambda_{e_i} \nabla_{w(t)} P_{e_i}(F(Z_t))$$

The question now is how to solve for the gradients corresponding to the enhancement and filtering constraints, $\nabla_{w(t)} P_{e_i}(F(Z_t))$ and $\nabla_{w(t)} P_{f_j}(F(Z_t))$ respectively. To accomplish this, we propose the concepts of model adaptation, feature adaptation, and signal adaptation. The idea being that we can separate these gradients into the three parts, each corresponding a different part of the sound modeling and detection pipeline. We can visualize these three components via the chain rule of derivatives shown in Equation 4.6, and summarize the three phases next.

$$\nabla_{w(t)} P_{e_i}(F(Z_t)) = \frac{\partial P_{e_i}(F(Z_t))}{\partial F(Z_t)} \cdot \frac{\partial F(Z_t)}{\partial Z_t} \cdot \frac{\partial Z_t}{\partial w(t)}$$

- **Model Adaptation:** $\frac{\partial P_{e_i}(F(Z_t))}{\partial F(Z_t)}$
- **Feature Adaptation:** $\frac{\partial F(Z_t)}{\partial Z_t}$
- **Signal Adaptation:** $\frac{\partial Z_t}{\partial w(t)}$

A visualization of the three phases are shown in Figure 4.1. The three typical steps for acoustic detection is highlighted in the forward pass, where the raw signal is preprocessed, features are computed, and the model is used to estimate the probability that the sound is present. To compute the full gradient with respect to our filter coefficients, $\nabla_{w(t)} P(F(Z_t))$, we take advantage of the chain rule of derivatives to compute gradients corresponding to each of these components for the
detector (model adaptation), features (feature adaptation), and the filtering process (signal adaptation), as shown in the backward pass of Figure 4.1. In the following subsections, we discuss each of these components in detail.

**Model Adaptation**

In model adaptation, we compute the gradient of the machine learning decision function. Through chain rule, model adaptation computes the gradient of the model decision function, $P_c(F(Z_t))$, for an acoustic detector for sound $c$, with respect to the features computed from the processed and filtered signal, $F(Z_t)$. In other words, we are "adapting" our filter coefficients based on the output to the detector with respect to the input.

Computing this quantity is possible for many different types of sound models and detectors. In general, the inputs to the classifier will be some set of features (i.e. MFCCs or even the raw frequency-domain signal) of dimension $n_F$. As such, model adaptation results in a row vector, $\frac{\partial P_c(F(Z_t))}{\partial F(Z_t)}$, of dimension $n_F$.

In this work, we use three different types of sound classifiers to show CIBF’s versatility: support-vector machine with radial basis function kernel (SVM RBF), random forest classifiers
(RF), and mixture of Gaussians (GMM). The model adaptation derivations for each of these classifiers are shown in the Appendix (Section A.0.1).

**Feature Adaptation**

The second step is feature adaptation, which corresponds to feature computation module, where we compute the gradient of the features, $F(Z_t)$, with respect to the filtered signal, $Z_t$. In general, the feature generation process reduces the dimensions of the filtered or raw signal. If the signal has dimension $n_B$ and the computed features have dimension $n_F$, then feature adaptation, $\partial F(Z_t)/\partial Z_t$ yields a gradient field of dimension $n_F \times n_B$.

Since most acoustic features mainly involve binning (i.e. weighting and summing bins between predefined frequencies), $\partial F(Z_t)/\partial Z_t$ is generally simple to compute. In this paper, we utilize two different acoustic features: mel-frequency cepstral coefficients (MFCC) and non-uniform binned periodogram (NBIP) [8]. The feature adaptation derivations for these features are shown in the Appendix. Note the simplest case of feature extraction is using no features at all (i.e. using the raw signal directly). In this case, feature adaptation yields an identity matrix for the gradient.

**Signal Adaptation**

The third step, signal adaptation, computes gradients of the filtered signal, $Z_t$, with respect to the filter coefficients, $w(t,f)$. In model adaptation, we computed $\partial F(Z_t)/\partial Z_t$ of dimension $n_F \times n_B$, where each row corresponds to one component of the computed feature and each column corresponds to one frequency bin of the raw signal. Now we must compute the gradient of with respect to each set of filters per frequency $f$.

The $p$-th column of our $\partial F(Z_t)/\partial Z_t$ matrix from feature adaptation corresponds to the $p$-th frequency bin’s gradient contribution to each of the $n_F$ feature bins. In other words, the $(l, p)$ entry of this matrix corresponds to the effect that only the $p$-th frequency has on the $l$-th feature bin. As such, to compute gradients corresponding to filters of the $p$-th frequency $f_p$, we only need to use the $p$-th column of $\partial F(Z_t)/\partial Z_t$. To do this, we multiply $\partial F(Z_t)/\partial Z_t$ by a column vector $C_p$, 102
whose $p$-th entry is 1 and all other entries are 0.

Acoustic detectors generally compute features based on the power spectrum. The power spectrum of the filtered signal is $S_f(t, f) = w^*(t, f)x(t, f)x^*(t, f)w^*(t, f)$. The full feature adaptation and signal adaptation output is shown in Equation 4.7. $y(t, f_p) = w^*(t, f_p)x(t, f_p)$ refers to the filtered signal at frequency $f_p$.

$$
\frac{\partial F(Z_t)}{\partial Z_t} \cdot \frac{\partial Z_t}{\partial w(t, f_p)} = \frac{\partial F(Z_t)}{\partial Z_t} \cdot C_p \cdot \frac{\partial S_f(t, f_p)}{\partial w(t, f_p)}
$$

$$
= \frac{\partial F(Z_t)}{\partial Z_t} \cdot C_p \cdot x(t, f_p)y(t, f_p)
$$

(4.7)

### 4.3 System

We build the AvA acoustic enhancement and filtering pipeline using CIBF as the centerpiece, allowing AvA to account for a wide range of features and models. CIBF leverages both content-based filtering (pre-trained sound models), and spatial filtering (multiple microphones). However, beamforming requires the direction of the source as input. In this section, we first introduce our localization module that detects and localizes significant sources in the environment. Then, we introduce the full AvA architecture.

#### 4.3.1 Acoustic Localization

To enable beamforming, we need to incorporate a localization module. The localization module needs to locate all the significant sources in the environment from different directions. Then, AvA will utilize CIBF to "beamform" to the direction of the sources and enhance/filter detected sources specified by the user or application.

There are numerous works that address multiple-source localization. In general, most algorithms scan across all directions where a potential source could be and compute a power response across all directions. The number and location of significant peaks in this curve are the number and estimated location of sources respectively. Each method differs in how they compute this power
response curve and how they search for peaks. Methods such as steered-response power (SRP) or steered-response power phase transform (SRP-PHAT) apply a time shift or phase shift and use generalized cross correlation between microphone pairs as the power response at each direction $d$ [142]. The idea is that signals coming from direction $d$ will be added constructively, while signals not aligned with direction $d$ will be destructively added (i.e. attenuated). Methods such as MUSIC and its variants [83] use eigenspace methods to compute a similar correlation metric. Frequency-domain versions of these methods generate power response curves for each frequency and aggregate them before searching for peaks. Generating these curves per frequency is expensive.

Instead, we utilize the method presented in [70]. This work, rather than computing a curve across each frequency, compares the observed phase differences between microphone pairs to the expected phase difference we would expect to see if a source was coming from a specified direction $d$ and assigns all the energy of the frequency to the direction where the source is most likely arriving from, which greatly reduces computation.

4.3.2 AvA Architecture

Figure 4.2 shows AvA’s full adaptive system architecture. The red dotted box highlights the content-informed adaptive beamforming module, while the green arrows and text highlights the detection and filtered signal outputs to AvA.

First, we sample a window from each of our $n_m$ microphones, $x_m(t, f)$ for $1 \leq m \leq n_m$. Then, we apply our filters, learned from previous iterations, to obtain $n_s$ different filtered sources, $Z_s^t$, for $1 \leq s \leq n_s$, at time window $t$. The number of sources present, $n_s$, and their corresponding direction of arrivals, $\theta_s^t$, is estimated by the localization module in the previous time iteration. Additionally, the update and apply filters module outputs the individual filtered microphone channels, $\hat{x}_m(t, f)$ for $1 \leq m \leq n_m$. We obtain $\hat{x}_m(t, f)$ for microphone $m$ by applying each filter onto each of the microphone channels, to diminish all sounds we want to filter out and enhance all sounds we want to retain.
Afterwards, the cleaned microphone channels are used in the localization module to estimate the number of significant sources in the environment, \( n_s \), and their location or direction of arrival, \( \theta_{t+1}^s \), that will be used to update source filters for the next time window, \( t + 1 \), as shown by the dotted arrow from the source detection and localization module back to the update and apply filters module. Additionally, the filtered sources, \( Z_t^s \), are used as inputs to the CIBF module, highlighted in the dotted red box, where sound analysis (acoustic detection), model adaptation, feature adaptation, and signal adaptation are performed to alter filter coefficients to enhance or reduce user specified sounds. These filters are applied at the next iteration and the cycle continues adaptively.

In the first iteration, AvA analyzes the raw audio channels; in other words, AvA’s initial filter starts off as all-pass, similar to traditional beamforming. Additionally, AvA may experience prob-
lems with convergence if the direction of the sound sources change too fast or randomly, just like in traditional beamforming. However, sound sources in most common applications generally move sufficiently slow. In Section 4.5, we demonstrate that AvA can adapt to an application in urban safety where sound sources (vehicles) move at tens of miles per hour.

4.3.3 Discussion

We note that in order to take full advantage of AvA, we require trained detectors to perform CIBF. However, AvA can also operate without any detectors or trained models. If no noise or target sound detectors are provided, AvA will perform LCMV beamforming (Equation 4.1) and only utilize spatial filtering.

Beamforming has often been compared to blind source separation (BSS). The primary difference is that beamforming filters signals by "steering" to a user-specified direction, whereas BSS does not require this input. At first glance, it would seem that BSS is more advantageous than beamforming in our application. However, many applications require the location of filtered signals (we explore one application in Section 4.5). Although BSS does not require the location or direction of sources as input, it also does not output source directions. Moreover, phase information critical to estimating source directions, which are present in raw signals, are not retained once BSS has been applied. As such, we take the beamforming-based two-step localize-then-filter approach, detailed in this section, to ensure we have source directions that are associated with filtered signals. Additionally, both beamforming and BSS utilize source direction found, in phase information between microphone channels, to perform filtering. As such, if two similar sounding sources appear from different directions, both BSS and our proposed CIBF method can differentiate the sources.

4.3.4 Integrating AvA into New Applications

A developer can integrate AvA into their own applications by providing up to three parameters. The first component is the relative locations of the microphone array that are needed in the tradi-
Table 4.1: Summary of evaluation scenarios and different configurations of AvA.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Detection Model</th>
<th>Features</th>
<th>Comparison Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target: Crying</td>
<td>Support vector machine (SVM)</td>
<td>Mel-frequency cepstral coefficients (MFCC)</td>
<td>LCMV Beamforming (AvA - LCMV) [71]</td>
</tr>
<tr>
<td>Noise: Construction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target: Dog</td>
<td>Random Forest</td>
<td>Non-uniform binned periodogram (NBIP)</td>
<td>Redress BSS (RBSS) [143]</td>
</tr>
<tr>
<td>Noise: Vehicle</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target: Piano</td>
<td>Gaussian mixture model (GMM)</td>
<td></td>
<td>Two Step Mask Learning (TDNN) [144]</td>
</tr>
<tr>
<td>Noise: Speechnoise</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target: Wild animals</td>
<td></td>
<td>Nonfiltered (NF)</td>
<td></td>
</tr>
<tr>
<td>Noise: Wind</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The second component is the sound models (optional) to use for filtering or enhancement, including the models themselves ($P_{e_i}$ and $P_{f_j}$) and the input signal representation, $F(\cdot)$. These models can be a wide range of detectors and could be models that the application itself would leverage. If sound models were provided for filtering or enhancement, the developers should also supply weight parameters, $\lambda_{e_i}$ and $\lambda_{f_j}$, which determine how much emphasis to place on enhancing or filtering specific sounds. Setting $\lambda_{f_j}$ higher than $\lambda_{e_i}$ would guide AvA to prioritize filtering out sound $f_j$ over enhancing sound $e_i$. To streamline this process, we provide developers common preset weights that they can choose.

### 4.4 AvA Evaluation

In this section, we evaluate the performance of AvA in various scenarios and configurations. The goal of AvA is to improve detection or "enhance" target sounds and degrade detection or "filter" out other noises that users can specify.

We look at four different scenarios where AvA may be useful. Additionally, we vary the model types and signal features used in each scenario to show the versatility of AvA. Table 4.1 summarizes the configurations we ran to evaluate the performance of AvA. The scenarios in which we evaluated AvA are described next.

- **Scenario 1: Baby crying enhancement in presence of urban construction sounds.** Parents need to know when their children are crying to take care of them, but loud construction noises could make this challenging. An audio-based child alert system may make use of AvA to filter out construction and enhance crying.
• **Scenario 2: Dog barking enhancement in presence of oncoming vehicles.** A dog barking or whimpering could be a sign of it requiring attention, but it could be difficult for an application to hear it in presence of urban and vehicle sounds. A pet care application, that uses audio to detect and alert caretakers of pet distress sounds, could benefit from AvA by filtering out urban sounds and enhancing pet sounds.

• **Scenario 3: Music enhancement in presence of speech and speechnoise.** In social gatherings, there may be music playing in the background that users may want to enjoy. An acoustic augmented reality application could enhance the music for the user and reduce the ambient speechnoise.

• **Scenario 4: Wild animal enhancement in presence of wind.** In a wildlife environment, a person may want to observe the sounds of animals or nature. However, the environment could be very windy and loud. A wildlife related application could enhance wildlife sounds and filter out wind sounds to improve the overall acoustic experience for users.

In each of these scenarios, we train a model for the sound we want to enhance (target) and the sound we want to filter out (noise) using AvA. The sound models and signal features used are also summarized in Table 4.1. In total, we evaluate AvA using three different types of detectors (SVM RBF, RF, GMM) with two different features (MFCC, NBIP), for a total of six configurations per scenario. We generate GMMs using a dirichlet process to automatically find the best number of clusters to use per model [145].

We compare AvA against other types of filtering methods, summarized in Table 4.1. The LCMV beamforming algorithm uses spatial differences between microphones to perform filtering [71]. We denote LCMV beamforming as AvA - LCMV because, as mentioned in Section 4.3.2, AvA directly performs LCMV beamforming if sound models are not provided. Redress BSS (RBSS) is a state-of-art blind source separation algorithm [143]. Two Step Mask Learning (TSML DNN) is a state-of-art deep neural network for sound source separation [144]. For each method, we filter our signals through the filtering method and evaluate detection performance using one of the
Table 4.2: Target and noise detection performance in scenario 1 (target: crying and sobbing + noise: construction).

<table>
<thead>
<tr>
<th>Detector Types</th>
<th>SVM RBF</th>
<th>Random Forest</th>
<th>Gaussian Mixture Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Target</td>
<td>Noise</td>
<td>Target</td>
</tr>
<tr>
<td></td>
<td>TP</td>
<td>FP</td>
<td>TP</td>
</tr>
<tr>
<td>SVM RBF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AvA</td>
<td>0.821</td>
<td>0.180</td>
<td>0.153</td>
</tr>
<tr>
<td>AvA - LCMV</td>
<td>0.789</td>
<td>0.184</td>
<td>0.423</td>
</tr>
<tr>
<td>RBSS</td>
<td>0.748</td>
<td>0.191</td>
<td>0.483</td>
</tr>
<tr>
<td>TSML DNN</td>
<td>0.723</td>
<td>0.182</td>
<td>0.276</td>
</tr>
<tr>
<td>NF</td>
<td>0.734</td>
<td>0.199</td>
<td>0.899</td>
</tr>
<tr>
<td>NBIP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MFCC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AvA</td>
<td>0.754</td>
<td>0.210</td>
<td>0.133</td>
</tr>
<tr>
<td>AvA - LCMV</td>
<td>0.735</td>
<td>0.230</td>
<td>0.376</td>
</tr>
<tr>
<td>RBSS</td>
<td>0.702</td>
<td>0.200</td>
<td>0.354</td>
</tr>
<tr>
<td>TSML DNN</td>
<td>0.713</td>
<td>0.229</td>
<td>0.234</td>
</tr>
<tr>
<td>NF</td>
<td>0.685</td>
<td>0.202</td>
<td>0.873</td>
</tr>
</tbody>
</table>

Table 4.3: Target and noise detection performance in scenario 2 (target: dog + noise: vehicles).

<table>
<thead>
<tr>
<th>Detector Types</th>
<th>SVM RBF</th>
<th>Random Forest</th>
<th>Gaussian Mixture Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Target</td>
<td>Noise</td>
<td>Target</td>
</tr>
<tr>
<td></td>
<td>TP</td>
<td>FP</td>
<td>TP</td>
</tr>
<tr>
<td>SVM RBF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AvA</td>
<td>0.897</td>
<td>0.137</td>
<td>0.163</td>
</tr>
<tr>
<td>AvA - LCMV</td>
<td>0.849</td>
<td>0.132</td>
<td>0.432</td>
</tr>
<tr>
<td>RBSS</td>
<td>0.832</td>
<td>0.136</td>
<td>0.456</td>
</tr>
<tr>
<td>TSML DNN</td>
<td>0.858</td>
<td>0.135</td>
<td>0.287</td>
</tr>
<tr>
<td>NF</td>
<td>0.812</td>
<td>0.133</td>
<td>0.909</td>
</tr>
<tr>
<td>NBIP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MFCC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AvA</td>
<td>0.911</td>
<td>0.132</td>
<td>0.123</td>
</tr>
<tr>
<td>AvA - LCMV</td>
<td>0.837</td>
<td>0.139</td>
<td>0.331</td>
</tr>
<tr>
<td>RBSS</td>
<td>0.819</td>
<td>0.137</td>
<td>0.298</td>
</tr>
<tr>
<td>TSML DNN</td>
<td>0.869</td>
<td>0.140</td>
<td>0.178</td>
</tr>
<tr>
<td>NF</td>
<td>0.800</td>
<td>0.131</td>
<td>0.886</td>
</tr>
</tbody>
</table>

detector types and signal features listed in Table 4.1. AvA directly uses these detectors to perform filtering and detection. As a baseline, we compare the filtering methods against the "nonfiltered" signals (NF), where we directly pass the raw signals into the detector. We generate the following datasets for evaluation.

- **Base dataset**: For each of the four scenarios, we extract 10 minutes of audio for both the sound we wish to enhance and the sound we wish to remove (80 minutes total). We extracted sounds from the Google Audioset dataset [86].

- **Mixed testing dataset**: This dataset contains mixtures of sounds from our different scenarios and is built from the base dataset. We use a six microphone uniform circular array (UCA), with a 15cm diameter, to record mixtures. In each scenario, we select a random clip from our
Table 4.4: Target and noise detection performance in scenario 3 (target: piano + noise: speech-noise).

<table>
<thead>
<tr>
<th>SVM RBF</th>
<th>Random Forest</th>
<th>Gaussian Mixture Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
<td>Noise</td>
<td>Target</td>
</tr>
<tr>
<td>TP FP</td>
<td>TP FP</td>
<td>TP FP</td>
</tr>
<tr>
<td>AvA</td>
<td>0.873 0.162</td>
<td>0.860 0.162</td>
</tr>
<tr>
<td>AvA - LCMV</td>
<td>0.849 0.159</td>
<td>0.839 0.167</td>
</tr>
<tr>
<td>RBSS</td>
<td>0.802 0.164</td>
<td>0.809 0.159</td>
</tr>
<tr>
<td>TSML DNN</td>
<td>0.852 0.158</td>
<td>0.819 0.164</td>
</tr>
<tr>
<td>NF</td>
<td>0.771 0.164</td>
<td>0.780 0.159</td>
</tr>
</tbody>
</table>

Table 4.5: Target and noise detection performance in scenario 4 (target: wild animals + noise: wind).

<table>
<thead>
<tr>
<th>SVM RBF</th>
<th>Random Forest</th>
<th>Gaussian Mixture Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
<td>Noise</td>
<td>Target</td>
</tr>
<tr>
<td>TP FP</td>
<td>TP FP</td>
<td>TP FP</td>
</tr>
<tr>
<td>AvA</td>
<td>0.849 0.160</td>
<td>0.830 0.163</td>
</tr>
<tr>
<td>AvA - LCMV</td>
<td>0.841 0.161</td>
<td>0.818 0.166</td>
</tr>
<tr>
<td>RBSS</td>
<td>0.808 0.161</td>
<td>0.777 0.159</td>
</tr>
<tr>
<td>TSML DNN</td>
<td>0.822 0.160</td>
<td>0.798 0.170</td>
</tr>
<tr>
<td>NF</td>
<td>0.754 0.158</td>
<td>0.766 0.161</td>
</tr>
</tbody>
</table>

target class (i.e., crying) and a random clip from the noise class (i.e., construction). Then, we play both sounds from two different speakers placed at random directions from the UCA. In this way, all the recordings are mixed in the real world rather than artificially, as is commonly done in many works. In total, we generate 30 minutes of mixtures for each scenario (2 hours total). The mean signal-to-noise ratio of the target sound for each scenario is listed next:

- **Scenario 1**: -6.6 dB
- **Scenario 2**: -5.4 dB
- **Scenario 3**: -3.2 dB

110
- Scenario 4: -4.7 dB

- **Training and testing datasets**: For each scenario, we have 50 minutes of audio (*base dataset* + *mixed dataset*). We take 80% of the audio and use them to train detection classifiers using the features and models listed in Table 4.1. We take the rest of the 20%, filter them using AvA and the comparison methods listed in Table 4.1, and use them to evaluate detection performance (results shown in Tables 4.2, 4.3, 4.4, 4.5).

To train the TSML DNN, we take random target sound clips and noise clips from the *base dataset* and artificially mix them together to use as inputs. We need to artificially mix these sources because DNN methods require the ground truth sources to compute loss functions. Recording a mixture in the real-world does not give us access to the exact ground truth sources, whereas artificially mixing signals directly uses the ground truth to create training data.

Tables 4.2, 4.3, 4.4, 4.5 show the detection performance metrics for the target sounds and noise sounds in the four scenarios after applying one of the filtering algorithms. For both target and noise detectors in each scenario, *we tune the detectors such that the false positive (FP) rates are relatively equal across all filtering methods in order to better visualize the improvement or degradation in true positive (TP) rate*. In all scenarios, across all filtering methods and detection models, AvA sees the largest increase in the true positive detection rate of our target sounds across all scenarios. Moreover, AvA also sees the largest decrease in detection rate of the noise signal (i.e. the signal that we want to attenuate) across all configurations and scenarios. This is because AvA intelligently leverages both spatial and content filtering to improve detection, while other methods leverage only one. Additionally, AvA directly optimizes over the detectors and features a user develops or supplies for detection. We would also like to highlight that the detection rates of the target sounds get enhanced while the noise sounds get diminished if we incorporate sound models (AvA) compared to only utilizing the spatial filtering portion of the system (AvA - LCMV). In Tables 4.2, 4.3, 4.4, 4.5, we highlight, in red, the best performing configuration (highest target true positive rate) for each of the scenarios. We also highlight in blue the configuration that yields the best noise filtering (lowest noise true positive rate). These values are summarized below along with
the target sound detection rate increase and noise detection rate decrease compared to no filtering (NF):

- **Scenario 1:**
  - Target: RF + MFCC (6.9% increase)
  - Noise: GMM + MFCC (78.9% decrease)

- **Scenario 2:**
  - Target: SVM RBF + NBIP (11.1% increase)
  - Noise: SVM RBF + NBIP (76.3% decrease)

- **Scenario 3:**
  - Target: GMM + MFCC (10.3% increase)
  - Noise: GMM + MFCC (60.8% decrease)

- **Scenario 4:**
  - Target: SVM RBF + MFCC (8.3% increase)
  - Noise: RF + MFCC (75.3% decrease)

This shows that each type of classifier or feature may perform better in certain scenarios. Being adaptable to a wide range of configurations is one of AvA’s greatest strengths over existing methods. AvA outperforms the methods we compared against because it leverages both spatial and data-driven filtering, improving the weaknesses of using just one type. Additionally, compared to deep learning, AvA is extremely flexible, requires less data, and does not require developers to dedicate large amounts of hardware and time to create new architectures specific to each new sound.

In this section, we showed AvA’s versatility and capability of improving detection for a wide range of user specified sounds in a variety of different scenarios. In the following sections, we take
a deeper dive into two real application scenarios: urban safety and audio privacy. In both applications, we integrate AvA into a real mobile/embedded platform, and compare the performance of the AvA-enhanced system against existing works in the respective area.

4.5 Case Study: Urban Safety

4.5.1 Background

Motor vehicle accidents are a growing concern. Since 2009, there has been more than a 50% increase in pedestrian motor vehicle fatalities in the United States, and more than 130,000 people are treated in hospitals for vehicle accident injuries per year [146]. Additionally, motor vehicle accidents are the first or second largest cause of work-related fatalities in every industry [64].

To improve urban safety, there have been several works that introduce acoustic wearables for detecting/localizing vehicles and alerting users to avoid accidents. [8, 11] introduce wearables and smartphone platforms that use an array of microphones and novel machine learning architectures to accomplish this. However, these works assume that vehicles will be the loudest sound in the environment and see degraded performance in noisy environments. [12] introduces a construction helmet wearable for construction worker safety. Since construction sites are generally very noisy, the authors propose an adaptive filtering architecture to filter out construction sounds to improve vehicle detection. However, this work requires the construction tool sounds to be modeled as a Gaussian mixture model using the raw magnitude spectrum as the signal representation. Additionally, this work needs a separate vehicle detection module later down the pipeline. AvA on the other hand can use a wide range of different sound detection models and can directly incorporate a vehicle detector to improve vehicle detection.

4.5.2 Integrating AvA into Acoustic Wearables for Urban Safety

We integrate AvA into an acoustic headset wearable that leverages an array of microphones. The system architecture for the AvA-enabled, real-time, urban safety wearable is shown in Figure 4.3. We borrow the embedded wearable platform from [8] and insert AvA as the preprocessing
and the vehicle detection module running in the smartphone system. If a vehicle is detected, an audio, haptic, and visual alert is sent to the user, which also shows the direction of the vehicle in relation to the user.

For our use case scenario, we had a user wear the AvA-enhanced wearable next to a street in a bustling urban city while speaking to someone on the phone. The pedestrian is focusing on his conversation through his headset and is much less likely to hear oncoming vehicles. Additionally, the loud conversation from the pedestrian makes it more difficult for any acoustic wearable to detect and localize vehicles over the speech. In this scenario AvA employs a vehicle detector to enhance and a speech detector to filter out the user’s conversation in order to improve vehicle detection. We compare the AvA-enhanced acoustic urban safety wearable against the PAWS state-of-art pedestrian safety wearable [8] and the CSafe construction worker safety wearable [12]. We adapt the CSafe system to filter out speech rather than construction sounds. For all systems, we adopt the PAWS random forest based vehicle detector. For CSafe and AvA, we generate a Gaussian mixture model speech detector through a dirichlet process by using 5 minutes of recorded speech from the user. Using recorded speech from the user is a reasonable way to generate a speech model since the acoustic wearable use recordings and learn a user’s speech pattern over time during his/her current or past phone conversations.

Table 4.6 shows the performance metrics of all three systems. Just as in Section 4.4, we tune each system such that the true negative rate for vehicle detection is similar for all systems we evaluate to better visualize the improvement in the true positive rate. We see that the AvA-enabled system has the highest true positive rate, followed by CSafe and PAWS. This means that the AvA-enabled system was able to detect the most number of windows where a vehicle is present. PAWS has the worst performance because it does not employ any method to deal with loud non-vehicle sounds (the phone conversation). Additionally, AvA is able to outperform CSafe because CSafe only has a module to filter out speech. AvA not only reduces the effect of speech, but also directly uses the vehicle detector to enhance signals and improve vehicle detection.

Table 4.7 shows the localization error of AvA, PAWS, and CSafe. We see that PAWS performs
Figure 4.3: AvA-enhanced urban safety wearable architecture. The embedded hardware platform is borrowed from [8]. AvA directly uses the results from the vehicle detector to determine when to alert the user.

Table 4.6: Performance metrics of vehicle detection of AvA compared to other state-of-art acoustic-based urban safety wearables, while user is having a phone conversation.

<table>
<thead>
<tr>
<th></th>
<th>True Pos.</th>
<th>True Neg.</th>
<th>False Pos.</th>
<th>False Neg.</th>
<th>Vehicles Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvA</td>
<td>0.866</td>
<td>0.974</td>
<td>0.026</td>
<td>0.134</td>
<td>15/15</td>
</tr>
<tr>
<td>CSafe</td>
<td>0.834</td>
<td>0.973</td>
<td>0.027</td>
<td>0.166</td>
<td>14/15</td>
</tr>
<tr>
<td>PAWS</td>
<td>0.729</td>
<td>0.982</td>
<td>0.018</td>
<td>0.271</td>
<td>11/15</td>
</tr>
</tbody>
</table>

Table 4.7: Localization error comparison between AvA and other state-of-art acoustic-based urban safety wearables, while user is having a phone conversation.

<table>
<thead>
<tr>
<th></th>
<th>Avg. Error (degree)</th>
<th>Std. Dev. Error (degree)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvA</td>
<td>12.97°</td>
<td>11.88°</td>
</tr>
<tr>
<td>CSafe</td>
<td>16.62°</td>
<td>10.71°</td>
</tr>
<tr>
<td>PAWS</td>
<td>27.25°</td>
<td>16.39°</td>
</tr>
</tbody>
</table>

much worse than AvA and CSafe because its localization module is affected by the phone conversation of the user. We see that AvA and CSafe have similar performance because of their ability to filter out the loud phone conversation that adversely affects vehicle detection and localization. This shows that AvA can improve other aspects of acoustic sensing, beyond detection, by selectively enhancing or filtering specific sounds.

Table 4.8 shows the latency breakdown and power consumption comparisons. We note that, AvA utilizes the same hardware pipeline as CSafe. As such, the hardware processing and power consumption of the embedded platform are equivalent. Although the algorithms employed by AvA requires slightly more time to execute than CSafe, we note that the difference is less than 10ms, and both systems still operate at real-time on the order of the average person’s reaction time (242ms
Table 4.8: Power consumption and latency comparison between an AvA-enhanced wearable and other state-of-art urban safety wearables. The total latency is the time it takes for each system to process one window of audio. The power consumption shows the current draw from each embedded platform powered by a 3.3V battery.

<table>
<thead>
<tr>
<th></th>
<th>AvA</th>
<th>CSafe</th>
<th>PAWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Proc. and Sampling</td>
<td>228ms</td>
<td>228ms</td>
<td>224ms</td>
</tr>
<tr>
<td>Algorithmic Processing</td>
<td>14ms</td>
<td>8ms</td>
<td>91ms</td>
</tr>
<tr>
<td>Total Latency (hardware + algorithms)</td>
<td>242ms</td>
<td>236ms</td>
<td>315ms</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>69.0mA</td>
<td>69.0mA</td>
<td>18.9mA</td>
</tr>
</tbody>
</table>

vs 236ms). PAWS requires much less power because its hardware platform utilizes an application-specific integrated circuit (ASIC) that significantly reduces power consumption, whereas CSafe and AvA utilize a higher power consumption microcontroller. In future work, we also aim to reduce power consumption by integrating an ASIC. However even in its current state, the AvA-enhanced wearable can still operate continuously for 14.5 hours off of two standard AAA batteries with 1000mAh capacity, which is more than enough for daily use.

4.6 Case Study: Audio Privacy

4.6.1 Background

The growth of mobile devices and wearables has enabled numerous applications that improve our daily lives. However, the readily available sensors on our smartphones and personal devices have also been causing a growing privacy concern. In 2019, the VRT NWS news outlet analyzed more than 1,000 recordings collected through Google Assistant applications and found that more than 10% of recordings were not prefaced with the "OK Google" command and should never have been recorded [94]. In 2017, The New York Times found that more than 1,000 smartphone applications used software that is known to collect TV viewership data by listening to TV sounds [95]. In this section, we show how AvA can improve acoustic privacy in mobile platforms.

First, we integrate AvA into a mobile sleep monitoring application. Mobile sleep monitoring applications use the microphone on the smartphone to detect, record, and analyze breathing sounds
Figure 4.4: AvA-enhanced platform for filtering out speech and preserving privacy in mobile applications. Unlike in urban safety, audio privacy applications save raw breathing sounds. In the AvA-enhanced systems, we save the filtered signals rather than the raw audio to preserve privacy.

as the user sleeps. These applications use a threshold-based detector, which will record anything that is loud enough for a microphone to sense, including privacy sensitive speech. In this application, we integrate AvA into our own sleep monitoring application, where we focus on enhancing breathing sounds to improve sleep detection while filtering out speech to enhance user privacy.

Second, we demonstrate how voice command applications can incorporate AvA as a preprocessing step to filter out speech that may be recorded without the proper command word. In this case, we want to "enhance" the command phrase (we use the "OK Google" phrase for this demonstration), while filtering out other speech.

For both systems, we create a mobile system with the architecture shown in Figure 4.4. Unlike in the urban safety application, we only use the single microphone channel available in most smartphones. We sample one window of audio (here we use 250ms windows), pass it through to our AvA filtering architecture that filters out speech (both scenarios) and enhances either the "OK Google" command or breathing sounds. The output in both scenarios is a saved audio stream, which we then analyze for speech intelligibility using the Google Speech-to-Text API [105] as a measure of privacy.

In the sleep monitoring application, we compare the benefits of AvA against PAMS [13]. PAMS uses models of speech to filter out speakers, much like AvA. However, PAMS can only run on mobile platforms, using a single microphone channel, while AvA is adaptable to systems with more microphones. Just as in Section 4.4, we tune each system such that the true negative rate for breathing detection is similar for all systems we evaluate for better comparisons.
Table 4.9: Proportion of words correctly identified, incorrectly identified, and undetected by Google Speech-to-Text. A lower rate of correctly identified words correlates to a more privacy-aware system.

<table>
<thead>
<tr>
<th></th>
<th>Correct</th>
<th>Incorrect</th>
<th>Not Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvA</td>
<td>16.7%</td>
<td>8.3%</td>
<td>75.0%</td>
</tr>
<tr>
<td>PAMS</td>
<td>18.4%</td>
<td>10.2%</td>
<td>71.4%</td>
</tr>
<tr>
<td>Sleep as Android</td>
<td>92.1%</td>
<td>1.4%</td>
<td>6.5%</td>
</tr>
</tbody>
</table>

4.6.2 Integrating AvA into Mobile Platforms for Sleep Monitoring

We compare the AvA-enhanced sleep monitoring system against the PAMS-enhanced system and the Sleep as Android sleep monitoring application [102]. We had 7 different volunteers speak one of 11 passages while we recorded their voice. We used these recordings to train our GMM speech model for both PAMS and AvA. We also use AvA to enhance breathing and sleep sounds. To do this, we trained a Radial Basis Function support vector machine (RBF SVM) using 5 minutes of sleeping and breathing sounds that we extracted from Google Audioset [86]. AvA uses this detector to enhance breathing sounds and perform breathing detection, while PAMS only uses this detector to detect breathing.

To generate our testing set, we had the same volunteers speak 10 different passages while playing one of 10 breathing and snoring clips through a speaker. All three systems then record, process, and save the clips. We run each saved clip through the Google speech-to-text API to measure speech intelligibility. Table 4.9 shows speech intelligibility metrics of the recorded sleep sounds, including the percentage of words correctly identified, incorrectly identified, and not detected. We see that Sleep as Android has the highest percentage of correctly identified words, which could spell a serious breach of privacy. We see that both PAMS and the AvA-enhanced systems have a much lower correctly identified rate and much higher incorrectly identified and undetected rates, meaning they were able to obscure and filter out much more speech and preserve privacy. However, even after improving privacy, the PAMS and AvA-enhanced applications still need to perform their original goals; that is, to detect and analyze breathing and other sleep sounds.
Table 4.10: Performance metrics for sleep breathing detection.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AvA</td>
<td>0.946</td>
<td>0.109</td>
<td>0.891</td>
<td>0.054</td>
</tr>
<tr>
<td>PAMS</td>
<td>0.891</td>
<td>0.112</td>
<td>0.888</td>
<td>0.109</td>
</tr>
<tr>
<td>Sleep as Android</td>
<td>0.986</td>
<td>0.944</td>
<td>0.056</td>
<td>0.014</td>
</tr>
</tbody>
</table>

Table 4.10 shows the performance metrics for sleep event detection. We see that Sleep as Android has the highest true positive rate because it uses a threshold-based detector. This means that if the sound is loud enough, it will detect and record audio. As such, Sleep as Android also has the highest false positive rate (i.e. if a person speaks when there is no breathing, Sleep as Android will still detect and record). On the other hand, we see that the false positive rate of both PAMS and AvA-enhanced systems is much lower at only a slight cost to true positive detection. Additionally, we see that the AvA-enhanced system has a significantly higher true detection rate than the PAMS-enhanced system. This is because AvA directly uses the sleep detector to improve the detection of sleep sounds, whereas PAMS is unable to do so.

To process one window of audio, the AvA-enhanced system takes 36ms, while PAMS takes 31ms. Although PAMS is slightly faster, **AvA comfortably runs in real-time, taking far less time than the sampling window to execute.**

4.6.3 Improving Command Phrase Privacy in Smart Audio Applications

In this section, we analyze how AvA can be applied to mobile and smart home applications that use voice commands. Generally, these applications listen until the command phrase is heard (i.e. "OK Google"), and then start recording and analyzing the audio to extract the voice command. However as mentioned at the beginning of this section, there have been many instances where these applications have recorded speech without the command phrase, which poses a privacy concern. In this scenario, we configure AvA to "filter" general speech, while "enhancing" just the "OK Google" command. If at any point the "OK Google" command is detected, then we turn off the AvA filtering pipeline and record the raw unfiltered audio. Otherwise, the filtered audio is saved.
Table 4.11: Speech intelligibility of the AvA enhanced command phrase mobile application in scenarios where a command phrase was present and not present.

<table>
<thead>
<tr>
<th></th>
<th>Correct</th>
<th>Incorrect</th>
<th>Not Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>not present</td>
<td>13.6%</td>
<td>16.5%</td>
<td>69.9%</td>
</tr>
<tr>
<td>present</td>
<td>94.3%</td>
<td>0.8%</td>
<td>4.9%</td>
</tr>
</tbody>
</table>

We used the same models for speech generated for the sleep privacy scenario in Section 4.6.2. We also generated a mixture of Gaussians model for the "OK Google" command by having each volunteer record the phrase 10 times each. Then, we had each volunteer speak 20 commands and recorded them with our mobile platform. Half of the phrases contained the "OK Google" command at the beginning, and the other half did not.

Table 4.11 shows the speech intelligibility metrics of the AvA-enhanced system when the command phrase is spoken compared to when the command phrase is not present. We see that the percentage of correctly identified words is much higher when the command word is spoken because the system turns off the filtering process when the command phrase is detected. On the other hand, when the command phrase is not detected, the system continuously filters speech, which drastically reduces speech intelligibility.

4.7 Discussion

Though we have shown that AvA can improve the detection of specific relevant sounds to an application, while filtering out targeted noises, there are several limitations to our architecture, which we detail in this section.

4.7.1 Dynamically Changing Sounds to Enhance and Filter

The first major shortcoming is that developers need to select the specific noise to filter or sound to enhance by specifying a model of a sound to detect, but there is no automatic method to switch between different noise or sound models. For example in a continuously running breathing
monitoring application, there could be a wide range of different types of sounds in the environment that could overpower breathing sounds, such as speech, wind, vehicle sounds, etc. These noises may not occur at the same time and the types of noises present may change over time.

Because content-informed adaptive beamforming (CIBF) is formulated and can incorporate multiple noises to filter out or target sounds to enhance, one way to circumvent this issue is to incorporate models for all the sounds and noises you foresee your application needing to accommodate. However, the more sounds you incorporate, the higher the computational requirements needed to run AvA; this is similar to other solutions, where a bigger and more complex model is needed to filter or classify a greater number of sounds. Endlessly adding additional sounds and models for AvA to account for does not scale well, especially in low-power and resource constrained systems. Additionally, it could be difficult to predict exactly all of the different types of overpowering sounds that may occur in the environment while a system is running for hours or days on end; an audio-based platform on a mobile wearable that is on a person, may see many environments, each with a unique acoustic profile, and may need to account for many different noises throughout the day. A mechanism that allows AvA to dynamically swap out or change noises to filter or sounds to enhance would greatly improve AvA’s ability to perform robustly in continuously operating systems. One way to accomplish this is to allow users to dynamically select the types of significant noises to filter out or sounds of interest to detect in the environment. However, this adds burden to the user of the application that AvA is augmenting.

To partially account for this limitation, we created an automated mechanism for dynamically selecting models of noises currently present in the environment and applied AvA along with this mechanism for dynamically selecting noises into an embedded application for monitoring breathing during sleeping and naps, called BuMA [147]. BuMA was designed in particular to monitor for sleep disorders that affect millions of people and is especially concerning for babies and their parents/caretakers.

Over the course of five years, researchers [148] discovered that people with sleep apnea had a considerably elevated risk of sudden cardiac death. The danger was most significant for those
aged 60 and older with moderate to severe sleep apnea (20 episodes an hour). When their oxygen saturation levels fell below 78%, the risk rose by an additional 80%. Additionally, those with severe sleep apnea have a two to fourfold increased risk of irregular cardiac rhythms compared to individuals without sleep apnea. Other researchers[149] found that some people with sleep apnea are more than 2.5 times more likely to pass away due to a heart attack between 12 a.m. and 6 a.m. than those without sleep apnea.

Sleep apnea also commonly occurs in infants. Small preterm infants are most likely to have infant sleep apnea. It sometimes occurs in larger preterm or full-term infants. During the first month after birth, sleep apnea occurs in 25% of infants who weigh less than 5.5 pounds. The risk increases to 84% for infants who weigh less than 2.2 pounds. One particular cause of concern for parents is sudden infant death syndrome (SIDS), where an infant passes away during sleep due to defects in the brain that controls breathing and waking up from sleep [150].

BuMA is an audio-based, non-contact, and real-time breathing monitoring system using a microphone array, built on top of the Raspberry Pi ecosystem. We take an audio-based approach because audio does not require good lighting (required for camera-based methods), is more privacy-sensitive, and does not require a device to be in contact with the user (which is commonly the case for many infant breathing and health monitoring applications). However, normal breathing sounds tend to be soft and low-energy, and there can be many noises in the home or nearby environment that could overpower that of breathing, such as passing vehicles, loud construction sounds, or speech. BuMA, which improves upon AvA by incorporating a multi-class noise detector to dynamically select different noises currently present in the environment for AvA to filter out. We evaluate BuMA in a case study and show that BuMA outperforms other state-of-art filtering schemes for improving breathing detection by up to 12%.

Figure 4.5 shows the system architecture of BuMA, which is made up of two major components: an embedded edge and a cloud component.

The embedded processing unit consists of an array of microphones attached to a processing unit that users can place near the sleeping person or clip onto the crib of a baby to monitor breath-
Because sounds of normal breathing are generally soft and low in energy, there are a wide range of other sounds, such as music, speech, or construction sounds, that could overpower the sound of breathing, making detection much more difficult. As such, to perform robust breathing detection on this embedded edge processing unit, we integrate AvA. To perform breathing detection, we extracted mel-frequency cepstral coefficients (MFCC), a common feature used in many audio applications. These features are then input into a Support Vector Classifier (SVC) with RBF kernel. To train this model, we extract 96 10-second clips of breathing sounds from the Google Audioset dataset [86], and 92 10-second random clips of non-breathing sounds. We use the same parameters and model type (SVC) to train sound detectors for each of three noises (construction, speech, and music). We build our edge processing unit on a Raspberry Pi 3B+ and use the Re-Speaker Circular Array as our microphone array [151]. This microphone array is a 6 element circular array. All components of our breathing detection pipeline are run on this platform.

However, there could be a wide range of noises in the environment, and it may not be enough to filter out one type of sound in the environment. For instance, at one point there could be construc-
Figure 4.6: Confusion Matrix of multi-class sound detector

tion noises in the background, and in the next instance, the construction may stop, but someone could be having a very personal conversation nearby. AvA does not inherently have a mechanism to change which type of sound to filter. To address this challenge, we incorporate a cloud processing component that periodically detects the presence of different types of noises in the environment. Then, the cloud processing component downloads and installs different models onto the edge processing unit to filter out present noises in the environment.

Ideally, BuMA should use sound models and detectors of specific noises currently present in the environment to filter them out using AvA. Because loud noises currently present in the environment could change over time, there needs to be a mechanism to select which noises to filter out. To accomplish this, we use a multi-class sound detector to detect what noises are present in the environment at regular intervals. Whenever a new noise is detected in the environment, then the sound model of the newly detected noise is used to filter it out.
Our multi-class sound detector is a gradient boosting classifier that classifies input sounds into 3 classes of potentially different noises that could be commonly found in or near a home environment: speech, music and construction. We extract mel-frequency cepstral coefficients (MFCC) to use as input features. We train our multi-class sound detector to dynamically swap out noise models using audio clips extracted from Google Audioset [86]. For each noise/sound class, we use 600 10-second clips extracted from Audioset, which totals 5 hours of audio. The confusion matrix of our detector is shown in Figure 4.6. The total accuracy of our multi-class sound detector is 93%.

We realize that there could be many other noises in the environment, beyond the classes mentioned in this work, and that this multi-class sound detector can be substituted by any multi-class sound detection method, such as [152], to dynamically determine what noises are present in the environment.

Because this multi-class detector is more complex and requires more computation than models and detectors of specific sounds, we move this detector to the cloud, rather than running it at the edge, as shown in the bottom half of Figure 4.5. We transmit one window of audio every 5 seconds from the edge to the cloud to analyze for the presence of noises. If a new sound is detected, the cloud processing unit alerts the edge processing unit to switch noise models.

We only transmit audio from the edge to the cloud periodically to reduce the amount of data being transmitted. Because much of this audio could be very privacy sensitive, by reducing the amount of data we transmit, we improve the overall privacy of the system by keeping most of the data and computation at the edge.

To evaluate BuMA, we recruited 3 volunteers and compared BuMA against several filtering schemes. Figure 4.7 shows our set up for collecting samples of breathing from our volunteers. Each volunteer laid down on a mattress in the supine position. We collected breathing samples from each volunteer when BuMA was at the side of the mattress and when BuMA was hanging above the mattress, level with the head of the volunteer. Having BuMA hang above the mattress simulates the case where we may want to embed our platform onto a dream catcher, which are commonly hung above baby cribs for children to play with. We also vary the distance BuMA is
Figure 4.7: Data collection set up. We placed BuMA either to the side or above the mattress, next to where a person would sleep. Then, we played an interfering noise (construction, speech, or music) from a speaker. In this figure, the distance between speaker and the person’s head is 150 cm, and BuMA is 30 cm away right above the person lying down on the mattress in the supine position.

away from the volunteer’s head from 5 cm to 30 cm. At each position, we played a random clip from Google Audioset [86], that is one of the 3 classes of sounds (construction, music, speech) our multi-class sound classifier detects to dynamically select noise models. At regular intervals of 10 seconds (each clip we played from Audioset is 10 seconds), we randomly play another clip from one of the other classes of noises. We play these clips 150 cm away and at a random angle $\theta$ from the person’s head, as shown in Figure 4.7. We varied $\theta$ from $-45$ to 45 degrees.

Figure 4.8 shows the accuracy of breathing detection versus the distance of the microphone array from the person’s head when the microphone array is above the mattress and to the side of
Figure 4.8: Accuracy vs. distance BuMA is away from the head of the person, when BuMA is above the mattress (top) and to the side of the mattress (bottom). The average SNR is listed above each distance.
the mattress. We plot the accuracy of BuMA compared to existing filtering algorithms, including LCMV beamforming [71] and the AvA filtering platform. We list the average signal-to-noise ratio (SNR) at each configuration. The noise model we select for AvA is the model of the class of the noise that is initially playing. For example, if a clip of construction sound is first playing, then the sound model we use to perform noise filtering is the model of construction noises. Though not explicitly shown, we tune the detectors such that the false positive rate is the same across all comparison methods, meaning all differences in accuracy between the methods is due to differences in the true positive detection rate.

We see that BuMA has the highest accuracy when the microphone array is within 20 cm when the array is at the side of the mattress, and when the microphone array is within 30 cm when the array is above the mattress. Past these points, the SNR is too low for our localization and filtering algorithms to significantly detect and enhance breathing. For example, when BuMA is placed to the side of the mattress at 30 cm, filtering does not significantly improve the accuracy of the system. The SNR when the microphone array is above the mattress is higher because the mouth and nose are directly pointing at the array, so the microphones are in the direct path of breathing. BuMA outperforms just normally applying AvA because AvA does not have a mechanism to dynamically switch noise models on the fly. BuMA outperforms the rest of the filtering algorithms because it incorporates both spatial and data-driven filtering, while other methods utilize one or the other.

When BuMA is 10 cm away and above the mattress, BuMA achieves a 73% accuracy, while not applying any filtering results in a 66%, which is a 12% improvement.

BuMA samples 500ms windows of audio with 250ms overlap from its microphone array that is attached to the edge processing unit. The full breathing detection pipeline runs on the edge processing unit in 21 ms, which is much less than 250ms.

At the cloud, we receive one window of audio every 5 seconds to determine if new noises are in the environment and that the edge processing unit needs to switch noise models. The cloud processing unit runs its multi-class noise detector to make this determination in an average of 9.44 ms and notifies the edge processing unit. The latency between when the edge processing
unit begins to transmit audio windows to the cloud and when the edge processing unit receives a response from the cloud unit (including processing the multi-class noise detector) is on average 18.35ms, which is much lower than 250ms, the amount of time between processing successive windows.

We have shown that incorporating an external, more complex, classifier to periodically and dynamically change the noises and models we use to filter out noises can improve the performance of AvA if the dominant overpowering noise changes over time. However, more work is needed to implement and realize a centralized platform containing a library of sounds that any application can utilize download onto their systems.

4.7.2 Characteristics of Sounds and Models

The second limiting factor of AvA is the "strength" of the models and detectors used to enhance and filter sounds. AvA's performance depends heavily on the sound models used. If the detector or sound model does not effectively detect the presence of the sound or noise to enhance or filter, then AvA's ability to improve the detection of target sounds and reduce the detection of target noises will decrease to the performance of traditional adaptive beamforming. In some cases, the performance of AvA will be worse than adaptive beamforming, especially in cases where the sound model incorrectly detects the presence of a sound and directs AvA to adapt coefficients, even though that sound is not present. Performing content-based filtering in this case would distort the signal.

The models we primarily integrated into AvA were small and simple, yet still powerful; this is to see the performance of AvA in settings where the application or system may be limited in computational and storage resources and may not have the capacity to incorporate larger and more powerful models. Although incorporating more robust models is key to improving the performance of AvA, it may not be possible to incorporate such robust models into certain systems. We are exploring methods to incorporate segmented architectures that leverage computing on the cloud to incorporate features from more complex models, such as the model and architecture for dynamically changing noises to filter introduced in Chapter 4.7.1.
This leads to the third limitation of AvA, which is filtering or enhancing sounds that are very similar in nature or structure. Sounds or noises that are very similar in nature may require more complex and robust models to differentiate. For example, an application may be interested in detecting the speech of only a specific person and ignoring the speech of other people. Although there are many differences in the speech signals between unique speakers, there are many similarities that may require a more powerful model to distinguish robustly. If the platform integrating AvA cannot store or run these models due to resource constraints, then the application may see limited improvements using AvA.

The fourth limitation of AvA is the signal-to-noise ratio of the target sound. Just like in any acoustic system, if the signal of the target sound is below the noise floor, then it is very difficult to recover the target sound. We experimented in many scenarios, showing improvements even in challenging environments ranging from -5 to -10 dB. Past this range, it is very difficult for even a person to hear the presence of the target sound, and we have seen limited improvements when applying AvA in these excruciating circumstances.
Conclusion

Despite the rapid growth in artificial intelligence and computing in recent decades, enabling our environments and machines to better sense, actuate, and understand the physical world, our homes, offices, cities, and environments are not yet fully intelligent and autonomous. Audio is one of the most important sensing modalities used in intelligent devices. This thesis explores how we can more easily embed acoustic intelligence into smart devices and objects we commonly wear or find around us to move towards more intelligent environments.

First, we introduced an audio wearable platform that detects and localizes vehicles before a collision to improve pedestrian safety. We proposed and introduced novel signal features, machine learning architectures, and algorithms to robustly detect and localize vehicles in real-time and on a resource-constrained mobile and wearable platform. We incorporated an application-specific integrated circuit to improve battery life to more than half a day, powered off of a coin cell battery, and introduced new audio-based localization algorithms that combine the physics of audio waves and data-driven training processes to achieve high-resolution location estimates, while being robust to noise. We demonstrated the practicality and performance of our wearable platform in real urban scenarios. This work introduces common frameworks, architectures, systems, and concepts that can be applied to a wider range of wearable platforms that jointly utilize embedded wearables in conjunction with mobile platforms to provide services to users.

Second, we adapt our audio-wearable platform for pedestrian safety to the application of improving construction worker safety. One of the major challenges of this problem is that construction sites are filled with sounds of power tools greater than that of oncoming vehicles, making
vehicle detection and localization exceptionally difficult. To address this challenge, we introduce an adaptive noise filtering architecture that combines both spatial filtering and content-based filtering in an adaptive feedback architecture, leveraging the strengths of physics-based and data-driven approaches, to enable real-time and robust construction tool filtering in a resource-constrained mobile and embedded system. We demonstrate improvements of our audio-wearable system, after incorporating our filtering architecture, in real construction site scenarios. This work highlights the possibilities of many other applications benefiting from audio filtering to improve the detection of specific sounds in the environment.

Finally, we present a common selective filtering framework that is adaptable to a wide array of mobile, embedded, and cyber-physical audio applications. This framework leverages an algorithm we proposed, called content-informed beamforming (CIBF), that can incorporate a wide array of different signal representations and sound model/detector types to filter out or enhance specific sounds depending on the applications. This platform enables developers and engineers to select specific sounds in the environment to enhance or filter out and allows them to select the best or preferred signal representations and model types for their application. We demonstrate that our platform can improve the detection of target sounds, while filtering out and reducing the detection of specified noises, in a wide array of different application scenarios and in real mobile and embedded systems. This platform ultimately enables us to more easily embed robust acoustic intelligence into real-time embedded, mobile, and cyber-physical systems to create more intelligent environments.

Our work lays the foundation for truly intelligent environments. Our environments are becoming smarter due to an ever-growing number of smart devices, but many of these devices are used independently from each other, which drastically limits the amount of intelligence found in current smart environments [1]. We envision three exciting avenues of research that can be built upon the work presented in this thesis: physics-informed deep learning for resource-constrained platforms; intelligent building environments that can automatically discover, utilize, and adapt to a wide range of sensors, actuators, and devices in the environment; intelligent and fully connected
This thesis presents a selective audio filtering platform for audio signals that combines the physics of audio signals along with data-driven machine learning models. In recent decades, deep neural networks have been shown to outperform many traditional machine learning and signal processing algorithms. However, deep learning generally requires a large amount of training data, computation, and model complexity, making it difficult to embed into resource-constrained environments. First, this thesis opens up new directions in how we can create *physics-informed deep neural networks*, that incorporate the physics of our sensing modalities into the training and design of deep neural networks, to create deep learning models that are still very robust, yet capable of running in more resource-constrained mobile and embedded platforms. By incorporating the known physics of our sensing modalities into the design of deep learning models, we can create more efficient neural networks that enable us to embed more robust intelligence into our environments.

Second, we envision a collection of *artificial intelligence for building environments*, like homes and commercial buildings, that can automatically discover, utilize, and adapt to a wide range of sensors, actuators, devices, and intelligence embedded into the environment to automatically learn and perform a wide range of useful tasks. Many home automation services require a person, with some technical background, to install devices and program logic flows for their specific task and specific environment; this is highly inefficient because there could be a large amount of tasks a person may want automated, and each environment may require a different setup. An artificial intelligence that can dynamically discover whatever resources and embedded intelligence happens to be available in the environment and automatically learn a wide range of automation services, would enable us to scale truly intelligent environments rapidly across the globe. The work presented in this thesis allows us to more easily embed intelligence into the environments around us, which enables artificial intelligence that realize truly intelligent environments.

Finally, we envision *fully connected, networked, and intelligent cities* that utilize resources across a diverse range of platforms (e.g., traffic infrastructure, vehicular systems, and pedestrian
wearables) to become healthier and smarter. Our work focuses on embedding intelligence into specific systems and applications, such as wearables for improving urban safety; these systems generally affect a single person or application. Scaling and utilizing intelligence we embed across an entire buildings, neighborhoods, and cities will bring the impact of our embedded intelligence from an individual to a population to the entire world.
References


[102] Urbandroid, Sleep as android (version 20200806), 2010.


145


Appendix A: Content-Informed Adaptive Beamforming Model Adaptation and Feature Adaptation

In this chapter, we derive the model adaptation and feature adaptation gradients for a few of the models and feature representations introduced in Chapter 4.2. In general, the model adaptation of any sound model can be incorporated into CIBF, as long as the decision boundary of the model is known. In general, most features and signal representations can also be incorporated.

A.0.1 Model Adaptation Gradients

This section details the computations required for model adaptation, \( \frac{\partial P(F(Z_t))}{\partial F(Z_t)} \), introduced in Section 4.2, for three different detectors: support vector machine with radial basis function kernel (SVM RBF), random forest (RF), and a mixture of Gaussians (GMM).

SVM with Radial Basis Function

For a kernelized SVM model, the decision function, \( P(G) \), is shown in Equation A.1.

\[
P(G) = \sum_{i=1}^{n} \rho_i k(G_i, G)
\]

\[
k(G_i, G) = exp(-\gamma||G_i - G||^2)
\]

Here, \( G_i \) refers to one of the training samples used to train the SVM, \( n \) refers to the number of samples used to train the model, and \( G \) is our input window feature that we wish to classify (i.e. the features computed on our input signal, \( G = F(Z_t) \)). \( k(\cdot, \cdot) \) is the RBF kernel, \( \gamma \) is a user tunable constant for the radial basis function, and the \( \rho_i \)'s are parameters that are learned during the training process. To perform model adaptation with an SVM RBF, we take the gradient of the decision function, \( P(G) \), with respect to the input, \( G \), shown in Equation A.2.
\[
\frac{\partial P(G)}{\partial G} = \sum_{i=1}^{n} 2\rho_{i} k(G_i, G) \gamma(G_i - G)^T
\] (A.2)

**Gaussian Mixture Model**

For a Gaussian mixture sound model, we use the probability density function as the decision function, shown in Equation A.3.

\[
P(G) = \sum_{i=1}^{n} a_i N(G|\mu_i, \Sigma_i)
\] (A.3)

Here, \(n\) refers to the number of clusters in the GMM, and \(N(\cdot|\mu_i, \Sigma_i)\) refers to the Gaussian probability distribution with mean \(\mu_i\) and covariance \(\Sigma_i\). \(a_i > 0\) are weighting parameters learned during the training phase. The model adaptation step follows in Equation A.4.

\[
\frac{\partial P(G)}{\partial G} = -\sum_{i=1}^{n} a_i N(G|\mu_i, \Sigma_i) [\Sigma_i^{-1} (G - \mu_i)]^T
\] (A.4)

**Random Forest**

A random forest detector uses a collection of \(T\) decision trees. Each decision tree contains a collection of nodes. A decision tree begins at the root node, which has two children nodes. The tree makes a decision based on the input window that is being classified. For instance, if the \(k\)-th dimension of our input is greater than some threshold \(\alpha\), then it will travel down one path. Otherwise, it will go down the other path. Eventually, it will arrive at a node that has no children (leaf node). Each leaf has a class associated to it (i.e. for a binary classifier, each leaf node is labeled a "0" if a sound is not detected, or a "1" if the sound is detected). Every input will eventually be classified into one of these leaf nodes. A random forest will have each of its \(T\) trees make a decision on whether the sound is detected and uses a majority vote to determine the final result (i.e. if more than half the trees detected the presence of the sound, then the random forest will also detect the sound).
Because random forests performs classification using explicit rules rather than an equation, it is
difficult to compute gradients and perform model adaptation. To create an equation-based decision
function for a random forest, we view the random forest model as a clustering algorithm rather
than as a decision tree.

For the $i$-th decision tree in a random forest of $T$ trees, there is a collection of $n^1_i$ nodes labeled
"1" (detected) and a collection of $n^0_i$ nodes labeled "0" (not detected). Each node, $j$, with label $k$,
has a collection of training samples, with mean $c^k_{i,j}$, that fall within the boundaries of the node. We
can create a decision function by finding the distance of an input window, $G$, between the means,
$c^k_{i,j}$, of each node $j$ in each tree $i$, as shown in Equation A.5.

$$P(G) = \sum_{i=1}^{T} \left[ \sum_{j=1}^{n^0_i} ||G - c^0_{i,j}||_2^2 - \sum_{j=1}^{n^1_i} ||G - c^1_{i,j}||_2^2 \right]$$  \hspace{1cm} (A.5)

Essentially by minimizing the distance of input $G$ to all nodes that belong nodes labeled "1"
while maximizing the distance to nodes labeled "0", we may be able to improve detection. The
model adaptation step, $\frac{\partial P(G)}{\partial G}$, follows in Equation A.6.

$$\frac{\partial P(G)}{\partial G} = \left[ \sum_{i=1}^{T} \left( 2 \sum_{j=1}^{n^0_i} (G - c^0_{i,j}) - 2 \sum_{j=1}^{n^1_i} (G - c^1_{i,j}) \right) \right]^T$$ \hspace{1cm} (A.6)

A.0.2 Feature Adaptation Gradients

This section details the computations required for feature adaptation, $\frac{\partial F(Z_t)}{\partial Z_t}$, introduced in
Section 4.2, for two different feature schemes: non-uniform binned periodogram (NBIP) and mel-
frequency cepstral coefficients (MFCC). In discussing feature adaptation for NBIP, we also discuss
computation for general binning schemes (i.e. summing all energies within a frequency range).

Non-Uniform Binned Periodogram

The NBIP feature evenly bins all frequencies below frequency $f_m$ into $a$ bins and all frequencies
above $f_m$ into $b$ bins. If the frequency domain representation of our signal has $B$ bins and bin
number \( m \) refers to frequency \( f_m \), then each NBIP feature at index \( i \) consists of summing up \( \Delta_l = \frac{m}{a} \) bins if \( i \leq m \) (lower half) or \( \Delta_h = \frac{B-m}{b} \) if \( i > m \) (upper half). The NBIP binning scheme, which produces a feature vector \( \mathbf{v} = [v_1, v_2, ..., v_{a+b}]^T \) from the power spectrum of the input signal \( \mathbf{Z}_t = [z_1, z_2, ..., z_B]^T \) (we refer to this \( \mathbf{Z}_t \) as the same \( \mathbf{Z}_t \) introduced in Equation 4.2), is shown in Equation A.7 [8].

\[
v_k = \begin{cases} 
\sum_{i=(k-1)\Delta_l+1}^{k\Delta_l} g(z_i), & \text{if } 1 \leq k \leq m \\
\sum_{i=m+(k-a)\Delta_h}^{m+(k-a-1)\Delta_h+1} g(z_i), & \text{otherwise}
\end{cases}
\]

(A.7)

\[ g(\cdot) = 20 \log_{10}(\cdot) \]

It follows that \( \frac{\partial F(\mathbf{Z}_t)}{\partial \mathbf{Z}_t} \) is a Jacobian matrix of dimension \( c \times B \), where \( c = a + b \) (Equation A.8).

\[
\left( \frac{\partial F(\mathbf{Z}_t)}{\partial \mathbf{Z}_t} \right)_{k,j} = \begin{cases} 
\sum_{i=(k-1)\Delta_l + 1}^{k\Delta_l} g'(z_j), & \text{if } 1 \leq k \leq m, \text{ and } (k-1)\Delta_l + 1 \leq j \leq k\Delta_l \\
\sum_{i=(k-a-1)\Delta_h + 1}^{(k-a)\Delta_h} g'(z_j), & \text{if } k > m, \text{ and } (k-a-1)\Delta_h + 1 \leq j - m, \text{ and } (k-a)\Delta_h \geq j - m \\
0, & \text{otherwise}
\end{cases}
\]

(A.8)

\[ g'(x) = 20 \log_{10}(e) \frac{1}{x} \]

If the \( j \)-th frequency bin is part of the sum used to generate the \( k \)-th feature bin, then the \((k, j)\) entry equals the gradient of a function \( g(\cdot) \) on the frequency bin. Since NBIP bins the periodogram, \( g(\cdot) \) converts the magnitude spectrum into the dB scale.
Mel-Frequency Cepstral Coefficients

MFCCs are a common acoustic feature, that transforms the input power spectrum, $Z_t$, as shown in Equation A.9.

$$F(Z_t) = \frac{1}{N} \cdot D \cdot \log(M \cdot Z_t) \quad (A.9)$$

$N$ refers to the number of samples in the window (i.e. the FFT size). $D$ is the discrete cosine transform matrix of dimensions $c \times c$, where $c$ is the number of filter banks employed in the MFCC (typically 12 or 13). $M$ is the $c \times B$ matrix of filter banks applied onto the input $Z_t$. The $\log(\cdot)$ operator applies the natural logarithm to all entries of the input matrix. Both $D$ and $M$ are static matrices that can be precomputed. The feature adaptation step for the MFCC feature is shown in Equation A.10.

$$\frac{\partial F(Z_t)}{\partial Z_t} = \frac{1}{N} \cdot D \cdot \text{diag}(M \cdot Z_t)^{-1} M \quad (A.10)$$

Here, the $\text{diag}(\cdot)$ operator takes the input vector and creates a diagonal matrix by placing all values along the diagonal. Since $M \cdot Z_t$ applies the filter banks $M$ onto our input signal $Z_t$, $M \cdot Z_t$ is a $c$ dimensional vector, so $\text{diag}(M \cdot Z_t)$ is a $c \times c$ diagonal matrix.